Sistemas Informáticos 2005-2006

MÓDULO DE INTELIGENCIA ARTIFICIAL PARA EL RISK

Realizado por:

Carlos Gonzalo Castellanos
Juan Luis Granero Pérez
Antonio Silva del Pozo

Dirigido por:
Pedro A. González Calero

Madrid, 28 de septiembre de 2006

Componentes del Grupo

Carlos Gonzalo Castellanos

Juan Luis Granero Pérez

Antonio Silva del Pozo

ÍNDICE

1. Descripción del proyecto

1.1. Propósito

La aplicación que queremos desarrollar consiste en un software multiusuario de un simulador del juego de estrategia Risk, añadiendo además al sistema la capacidad de jugar el mismo con distintos niveles de inteligencia artificial.

The application that we want to develop consists of a multiuser software of a simulator of the game of Risk strategy, adding in addition to the system the capacity to play he himself with different levels from artificial intelligence.

1.2. Vista General

El proyecto consiste en una aplicación de simulación al juego de mesa denominado Risk. Se trata de conseguir jugar a este juego a través de un ordenador dando la posibilidad de que varios jugadores se enfrenten entre si, ya sean estos jugadores humanos o máquinas.

Este juego consiste en realizar un objetivo, que se es repartido al inicio de la partida, antes de que lo consiga otro jugador. Para ello habrá que llevar distintas tácticas a cuerdo con el objetivo del que se posea para que este sea cumplido en el menor tiempo posible.

The project consists of an application of simulation to the denominated dinner service Risk. One is to be able to play this game through a computer being given the possibility that several players face to each other, or are these human players or machines. This game consists of making an objective, that has been distributed to the beginning of the game, before another player obtains to it. For it it will be necessary to take different tactics from prudent with the objective of which it is controlled so that this it is fulfilled in the smaller possible time.

Al inicio de la partida se elegirán el número de jugadores que van a participar en la misma y decidiendo también qué numero de estos son máquinas. A continuación se sortean los objetivos. Una vez hecho esto se asignarán automáticamente los territorios a cada jugador y se les da la posibilidad de repartir los batallones en sus territorios a los jugadores humanos. En caso de ser máquina esto se hará de forma automática.

Durante la ejecución de la partida se irá controlando el turno de cada jugador y se le dará la posibilidad de que realice las tres posibles fases: **refuerzo** (colocar nuevos batallones en sus territorios), **ataque** (intentar conquistar aquellos territorios más convenientes) y **maniobra** (movimientos de ejércitos entre territorios colindantes que pertenezcan al jugador en cuestión).

1.3. Características Principales

- El modelo de la aplicación es vista controlador. Los clientes serán los jugadores humanos, que podrán crear partidas a su gusto, dentro de las posibilidades existentes. El sistema por su parte proporcionará al cliente todos los datos necesarios para que pueda refrescar la información de sus paneles y pantallas.
- Se dispone de un sistema, encargado de crear, gestionar y monitorizar distintas partidas, manejando a los jugadores máquinas.
- La interfaz gráfica de la aplicación debe disponer de toda la información necesaria para el desarrollo normal de una partida, esto es visualizar el tablero, indicando en todo momento quien es el propietario de cada territorio y el número de batallones que lo defienden. Además se podrá consultar en todo momento el objetivo del jugador humano que en ese momento sea poseedor del turno y también se podrá consultar las cartas risk que haya obtenido a ese momento.

2. Captura de Requisitos

2.1. Requisitos Funcionales

El objetivo fundamental de este proyecto es conseguir una aplicación que simule una partida al juego de mesa risk, dando la posibilidad de enfrentarse al ordenador y que este posea distintos niveles de inteligencia artificial proporcionado por diversas tecnologías. *Para más detalle, consultar apéndice C.*

TABLERO

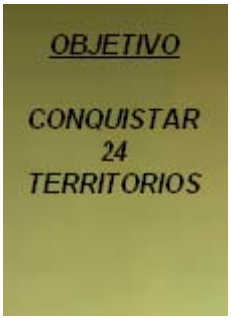
- El diseño está basado en el propio tablero del juego de mesa risk para proporcionar una sensación mayor de realismo.
- Los ejércitos serán visualizados en el tablero, el cual representa un mapa de los 6 continentes divididos por territorios o países. Según sea el propietario de dicho territorio, los ejércitos se ilustrarán del color específico de ese propietario.
- El reparto de los territorios entre los participantes de la partida se hará de forma aleatoria y equilibrada.

SISTEMA

- Será el encargado de gestionar todo para que la simulación siga su curso normal. Tras indicarle por consola de cuántos jugadores consta la partida y qué tipo de jugadores existen (humanos o máquinas) , irá indicando al jugador que posea el turno en la fase de ataque o refuerzo en la que esté, controlando que no realice acciones ilegales y de obedecerle en las decisiones que tome.
- Gestionará toda acción de los jugadores máquinas, ocultando dichos movimientos o ataques y mostrando en el tablero el resultado final. En una consola de eventos se indicarán dichas acciones para que el jugador humano sea consciente de los hechos..

OBJETIVOS

- Parte fundamental de este proyecto es saber cuando un jugador ha ganado la partida; dicha parte viene determinada por los objetivos. El sistema estará siempre comprobando los objetivos de todos los jugadores, para que en el momento que alguno lo consiguiera, indicar que la partida ha finalizado y dar a conocer el jugador ganador.



- En un principio, se implementará 3 tipos de objetivos, el primero es el de conquistar 24 territorios, el segundo consiste en destruir a un enemigo determinado y el tercero en conquistar 2 continentes previamente determinados.
- En el caso de que la partida conste de sólo 2 jugadores se procederá a la conquista del mundo por parte de un jugador, o lo que es lo mismo destruir al enemigo.

JUGADORES

- Son los usuarios activos de la aplicación. Se precisa que conozcan el funcionamiento del juego y su modo de operación.
- Una vez que el controlador se encuentre en la aplicación, éste podrá consultar su carta objetivo y sus cartas de refuerzo en cualquier momento de la partida.
- El sistema siempre estará vigilando a los jugadores las acciones que realicen, indicando por medio de avisos los posibles fallos o maniobras ilegales de los usuarios.
- Los jugadores podrán ser controlados por el sistema o bien jugadores humanos, dando la posibilidad de cualquier combinación de los mismos, siempre y cuando sean como mucho seis jugadores a la vez.
- En todo momento, el jugador humano es dueño de todas sus acciones, el sistema se limitará a indicarle en qué fase del juego está y esperará órdenes del mismo para atacar, defender o reforzar.

TERRITORIOS

- Son el elemento estrella de nuestro tablero. Representan la unidad de ataque, ya que los ataques se realizan desde un territorio a otro. El mapa está constituido por 42 territorios repartidos por los 6 continentes.

- Cada territorio posee un nombre y pertenece a un continente. Al inicio de la partida se visualizan vacíos, pero una vez empezada la misma, son rellenados por los ejércitos de su propietario al cual distinguimos por los colores de los mismos, único y exclusivo para cada jugador.
- En ningún momento de la partida un territorio podrá quedarse sin propietario.

CONTINENTES

- Representan agrupaciones de territorios. Estos forman parte de los objetivos, ya que la mayoría de ellos exigen la conquista de dos continentes en su totalidad y al mismo tiempo para conseguir la victoria de la partida.

- En principio, en nuestro tablero visualizamos los continentes reales, pero se da la posibilidad de la introducción de nuevos escenarios para la batalla.

PANEL DE INFORMACIÓN

- Se dispondrá de un panel informativo que proporciona al jugador humano de manera continuada, información relevante de la partida, como: los ataques realizados por los demás jugadores, refuerzos y maniobras.

INTELIGENCIA

- Se dispondrá de un sistema totalmente autónomo, encargado de llevar las acciones de los jugadores máquinas. El modo por el cual dicho tipo de jugadores

son guiados corresponderán a distintas implementaciones de inteligencia artificial.

- En función del tiempo del que se disponga, se irán introduciendo nuevos tipos de inteligencia, pero en principio los tipos estipulados de inteligencia son aquellos dirigidos por un sistema experto y por una red neuronal, dejando los algoritmos evolutivos para una posible ampliación.

CARTAS RISK

- Cada vez que un jugador entre en su fase de maniobra, habiendo conseguido conquistar al menos un territorio durante la fase de ataque, se le adjudicará una carta risk. Existen tres tipos de estas cartas: caballería, infantería y artillería.

- En función de la combinación de las distintas cartas risk el jugador que las canjee obtendrá un número determinado de batallones adicionales en su fase de refuerzo. Para poder canjear, un jugador debe poseer tres cartas del mismo tipo o bien una de cada. El número de batallones asignado es el siguiente:

3 CARTAS INFANTERÍA ===> 3 BATALLONES
3 CARTAS CABALLERÍA ===> 5 BATALLONES
3 CARTAS ARTILLERÍA ===> 7 BATALLONES
3 CARTAS DISTINTAS ===> 10 BATALLONES

2.2. Requisitos no funcionales

2.2.1. Requisitos de usuario

- Conocimiento de las reglas del juego.
- Conocimiento de la ejecución del juego.

2.2.2. Requisitos de diseño

- Uso de la misma metodología y procedimientos que en el juego de mesa.

- Fácil de modular para poder realizar pruebas periódicas sin alterar el desarrollo de la aplicación.
- Fácil ampliación para posibles mejoras

2.2.3. Requisitos del sistema

- Disponer de la Máquina Virtual de Java, JDK al menos 1.4.2.
- Intel Pentium II (AMD K6) o superior.
- Al menos 10 Mb. de espacio de disco duro.

2.3. Funcionalidad y Alcance

- Nuestro sistema se encargará de posibilitar una partida al juego de estrategia Risk, con distintos niveles de inteligencia artificial y con el mayor nivel de realismo.
- El proyecto se limitará a una interacción del usuario con distintos adversarios controlados por inteligencia artificial y con la posibilidad de enfrentarse a otros jugadores también humanos.
- Inicialmente se comenzará con la implementación de jugar entre jugadores humanos. Una vez cumplido esto, se añadirán jugadores máquina controlados por un sistema experto. Por último se incluirá un nuevo tipo de inteligencia artificial controlada por redes neuronales.

2.4. Tecnologías

- Se programará en lenguaje C++.
- Entorno de programación Borland C++Builder versión 5 o superior.

2.5. Ampliaciones

Estas ampliaciones se realizarán siempre y cuando los plazos de entrega lo permitan. Quedando anuladas si no pudieran cumplirse los mismos.

- **Introducción de la posibilidad de jugar en red.**

  Un aspecto bastante útil y adecuado a las nuevas tecnologías, sería añadir la posibilidad de que los distintos adversarios de una misma partida no se encontraran delante de un mismo ordenador, sino que cada uno desde su lugar de residencia pudieran enfrentarse entre sí.

  Para añadir dicha funcionalidad sería necesario que los usuarios estuvieran conectados a la misma red (esta puede ser INTERNET), un ancho de banda de dicha red lo suficientemente grande como para soportar la comunicación necesaria entre los adversarios y que la rapidez de ejecución de la partida sea aceptable.

Se dispondría de un servidor encargado de gestionar toda la comunicación entre los usuarios del sistema, así como el continuo refresco de las acciones de los jugadores a los que les correspondiera el turno.

- **Añadir nuevos tipos de inteligencia artificial.**

  Hoy en día, existen muchas tecnologías que permiten la implementación de sistemas inteligentes. Por tanto podríamos utilizar cualquier herramienta que actualmente exista o bien que tenga que venir todavía para la introducción de jugadores máquinas que desplieguen dichas herramientas.

- **Introducción de nuevos mapas**

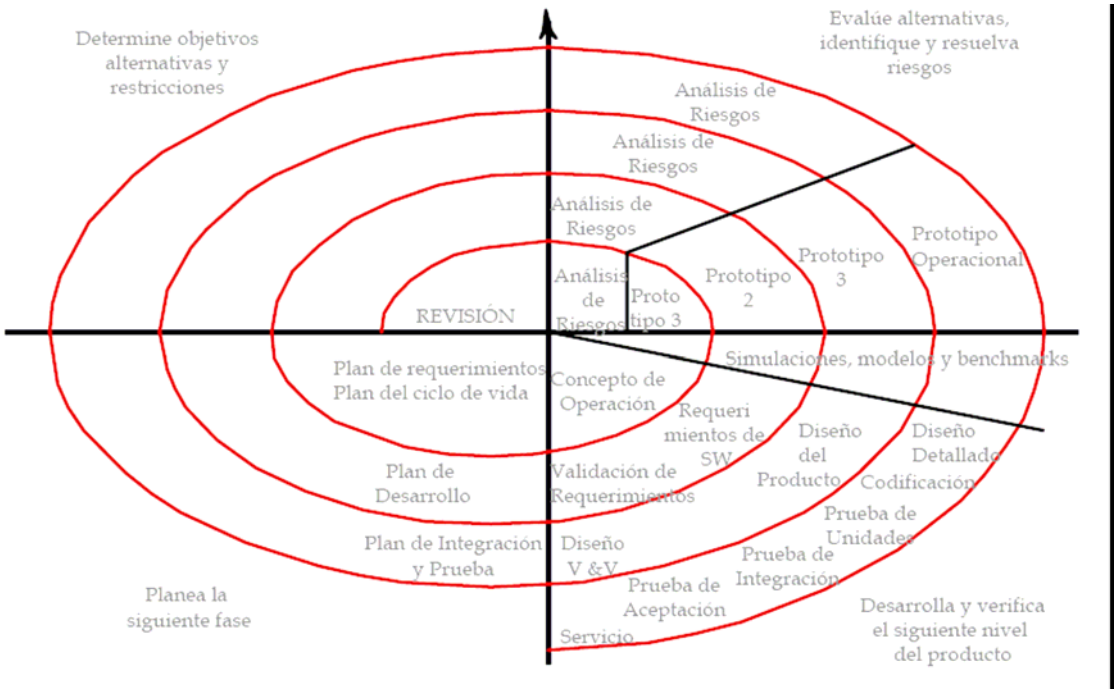
  Se introduciría la posibilidad de representar nuevos escenarios de la batalla en acorde con los tiempos actuales en los que nos movemos.

- **Introducción de sonidos y voces al sistema.**

  Determinadas situaciones de la aplicación se acompañarán con sonidos adaptados al tipo de la misma. Dando una sensación mayor de realismo acercando un poco más el ambiente en un escenario de guerra.

3. Plan de fase

Hemos elegido el modelo en espiral pues tiene la ventaja de que el análisis de riesgos se hace de forma explícita y clara. Las actividades de este modelo son una espiral donde cada vuelta nos supone una iteración.



Dado que tenemos un plazo fijo de entrega consideramos las fechas de cada iteración inamovibles y por eso las actividades no están fijadas del todo a priori, sino que las siguientes se eligen en función del resultado del análisis de riesgo, comenzando por el bucle interior.

3.1. Plan de fase original

Nuestro diseño original está compuesto por las siete iteraciones aquí descritas:

Iteración 0:

Fecha: 12 de Octubre de 2005.
Objetivos: Tener toda la documentación necesaria sobre el juego de estrategia, y especificación global del proyecto (requisitos, limitaciones, funcionalidades...).
Riesgos a solucionar: Desconocimiento de área, desconocimiento de nuevas tecnologías, habiendo mirado ya: qué y cómo lo vamos a usar.

Iteración 1:

Fecha: 20 de noviembre de 2005.

Objetivos: Conseguir un diseño inicial, compuesto de los casos de uso, tarjetas CRC y diagramas UML.

Riesgos a solucionar: Eliminar los riesgos tecnológicos (desconocimiento), realizando una pequeña prueba de cada uno de ellos. Preguntar y descubrir qué herramientas tenemos en el laboratorio y qué restricciones se nos presentan.

Iteración 2:

Fecha: 10 de diciembre de 2005.

Objetivos: Conseguir un diseño consistente y obtener una estructura estable de toda la arquitectura del proyecto.

Riesgos a solucionar:. Aprender a realizar estructuras preparadas para proyectos de gran magnitud.

Iteración 3:

Fecha: 15 de Enero de 2006.

Objetivos: Conseguir una versión inicial que posibilite una partida entre dos jugadores humanos.

Riesgos a solucionar: Debido a la coincidencia con los exámenes de febrero, riesgo de ausencia habitual, y compaginación de otras asignaturas, tener un control de la escasez de tiempo.

Iteración 4:

Fecha: 15 de abril de 2006.

Objetivos: Ampliación a la totalidad de las opciones de juego que posibilita una partida, hasta 6 jugadores y se introducen los distintos tipos de inteligencia.

Riesgos a solucionar: Introducción de los jugadores máquina en el sistema, acople en el desarrollo de la partida de agentes controlados por el sistema.

Iteración 5:

Fecha: 30 de mayo de 2006.

Objetivos: Pruebas del sistema en todos los aspectos, en caso de ser satisfactorias continuar con las ampliaciones.

Riesgos a solucionar: Resolver posibles conflictos que surjan una vez que se tiene el sistema completo y sea probado por personas externas al proyecto.

Iteración 6:

Fecha: 20 de Junio 2006.

Objetivos: Tener un proyecto estable y fiable.

Riesgos a solucionar: Todo lo que impida el correcto funcionamiento, que no de tiempo a terminar.

3.2. Evolución del plan de fase (FINAL)

En consecuencia de los problemas surgidos durante todo el proceso de desarrollo, el plan de fase tuvo que modificarse, añadiendo tres nuevas iteraciones. El plan quedó de la siguiente manera:

Iteración 0: (cumplida)

Fecha: 12 de Octubre de 2005.
Objetivos: Tener toda la documentación necesaria sobre el juego de estrategia, y especificación global del proyecto (requisitos, limitaciones, funcionalidades...).
Riesgos a solucionar: Desconocimiento de área, desconocimiento de nuevas tecnologías, habiendo mirado ya qué y cómo lo vamos a usar.

Iteración 1:

Fecha: 20 de noviembre de 2005.
Objetivos: Conseguir un diseño inicial, compuesto de los casos de uso, tarjetas CRC y diagramas UML.
Riesgos a solucionar: Eliminar los riesgos tecnológicos (desconocimiento), realizando una pequeña prueba de cada uno de ellos. Preguntar y descubrir qué herramientas tenemos en el laboratorio y qué restricciones se nos presentan.
Resultado: No se realizan las acciones aquí señaladas, se consigue un esbozo de una posible secuencia de las acciones del sistema (no se llegaron a plasmar con los casos de uso).

Iteración 2:

Fecha: 10 de diciembre de 2005.
Objetivos: Conseguir un diseño consistente y obtener una estructura estable de toda la arquitectura del proyecto.
Riesgos a solucionar:. Aprender a realizar estructuras preparadas para proyectos de gran magnitud.
Resultado: diagramas UML no realizados por falta de diseño estable de aplicación. Se aplazan por tanto a la siguiente iteración. Los casos de uso son realizados al igual que las tarjetas CRC.

Iteración 3:

Fecha: 15 de Enero de 2006.
Objetivos: Conseguir una versión inicial que posibilite una partida entre dos jugadores humanos.
Riesgos a solucionar: Debido a la coincidencia con los exámenes de febrero, riesgo de ausencia habitual, y compaginación de otras asignaturas, tener un control de la escasez de tiempo.

Resultado: Se consigue un diseño completo del proyecto con diagramas UML incluidos. Se pospone la versión inicial hasta después de exámenes.

Iteración 4:

Fecha: 15 de abril de 2006.

Objetivos: Ampliación a la totalidad de las opciones de juego que posibilita una partida, hasta 6 jugadores y se introducen los distintos tipos de inteligencia.

Riesgos a solucionar: Introducción de los jugadores máquina en el sistema, acople en el desarrollo de la partida de agentes controlados por el sistema.

Resultado: se consigue una versión inicial que posibilita una partida entre dos jugadores humanos. Se pospone para la siguiente iteración la ampliación de las demás funcionalidades.

Iteración 5:

Fecha: 30 de mayo de 2006.

Objetivos: Ampliación a la totalidad de las opciones de juego que posibilita una partida, hasta 6 jugadores y se introducen los distintos tipos de inteligencia.

Riesgos a solucionar: Adaptación del sistema a las nuevas tecnologías implementadas.

Resultado: Se consigue la posibilidad de jugar hasta seis jugadores y se introduce el jugador controlado por un sistema experto. Dicha sistema experto no ha sido implementado en su totalidad.

Iteración 6:

Fecha: 20 de junio de 2006.

Objetivos: Tener un proyecto estable y fiable.

Riesgos a solucionar: Se desarrolla por completo el sistema experto y se depuran algunos fallos. Debido a la falta de tiempo por problemas de planificación y observando el resultado del proyecto obtenido se decide prolongar la fecha de entrega hasta el 30 de septiembre introduciendo tres nuevas iteraciones.

Iteración 7:

Fecha: 20 julio de 2006.

Objetivos: Conseguir un sistema estable y fiable. Introducción de un nuevo tipo de inteligencia artificial controlado por una red neuronal.

Riesgos a solucionar: Errores internos difíciles de detectar y adaptar el proyecto a la nueva tecnología.

Resultado: Se obtiene un sistema estable pero se encuentran problemas con la red neuronal. Se aumenta el plazo de adaptación de la nueva tecnología.

Iteración 8:

Fecha: 10 de septiembre de 2006.

Objetivos: Conseguir un sistema estable con la red neuronal ya acoplada al sistema. Pruebas del sistema.

Riesgos a solucionar: Dificultad de implementación de la red neuronal.

Resultado: Se realiza una versión completa del sistema, pero no queda tiempo para realizar pruebas del sistema.

Iteración 9: (cumplida)

Fecha: 30 de septiembre de 2006.

Objetivos: Entrega del proyecto libre de errores y totalmente estable, con el sistema experto y la red neuronal integrados en el sistema.

Riesgos a solucionar: Errores difíciles de detectar.

4. Casos de uso

Nos describen la forma en que los actores interactúan con el sistema. Recogen fragmentos de funcionalidad que el sistema ofrece para aportar un resultado de valor para sus actores. Nos especifican una secuencia de acciones que el sistema puede llevar a cabo interactuando con éstos, incluyendo alternativas dentro de la secuencia

4.1. Actores que intervienen en la aplicación

- **Actor Cliente:** son los usuarios controladores que interactúan con la Aplicación Cliente. Pueden ser de dos tipos: humanos o máquinas.

- **Actor Sistema:** es el motor de la aplicación. Realiza toda la gestión de la partida.

Para cubrir la funcionalidad del sistema se han diseñado 11 casos de uso.

Caso 1	**Inicio partida**	
Objetivo en contexto	**Un jugador inicia la partida**	
Entradas		
Precondiciones	La aplicación se ha iniciado correctamente	
Salidas		
Poscondición si éxito	Se crea una nueva partida	
Poscondición si fallo	Reinicio de la aplicación	
Actores	Jugador/Sistema	
Secuencia Normal	**Paso**	**Acción**
	1	El jugador inicia la partida Indica número de jugadores y el tipo de los mismos (humano o máquina)
	2	El sistema crea una partida con el número indicado. Si error S-1

Secuencias Alternativas	**Paso**	**Acción**
	S-1	La partida no se puede crear, se procede al reinicio de la aplicación.

Caso 2	**Inicio Ataque**	
Objetivo en contexto	Elección de los territorios que participan en una batalla	
Entradas		
Precondiciones	Ha finalizado la etapa de refuerzo	
Salidas		
Poscondición si éxito	Los dos territorios que realizarán la batalla están elegidos	
Poscondición si fallo	Se procederá a repetir la acción.	
Actores	**Jugador/sistema**	
Secuencia Normal	**Paso**	**Acción**
	1	Se selecciona el territorio desde el cual se ataca. Si error S-1
	2	Se selecciona el territorio al cual se quiere atacar. Si error S-2
	3	Se selecciona el número de ejércitos con el que se desea atacar. Si error s-3
	4	Se inicia la batalla.
Secuencias Alternativas	**Paso**	**Acción**
	s-1	El territorio no pertenece al jugador, se procede a elegir otro territorio.
	s-2	El territorio pertenece al jugador, se procede a elegir otro territorio.
	s-3	No hay batallones suficientes para atacar, se vuelve a 1.

Caso 3	**Ataque simple**	
Objetivo en contexto	Transcurre la batalla	
Entradas		
Precondiciones	La batalla se inicio correctamente	
Salidas		
Poscondición si éxito	Se realiza la batalla.	
Poscondición si fallo	Se procede a reiniciar la batalla.	
Actores	Sistema	
Secuencia Normal	**Paso**	**Acción**
	1	Se consulta numero de ejércitos atacante y se lanzan dados rojos. Si error S-1
	2	Se consulta numero de batallones defensores y se lanzan dados azules. Si error S-2
	3	Se comparan dados.
	4	Se restan los batallones perdidos en la batalla.
	5	Se actualiza tablero

Secuencias Alternativas	Paso	Acción
	s-1	Se lanzan los dados rojos de nuevo.
	s-2	Se lanzan los dados azules de nuevo.

Caso 4	Ataque hasta el final	
Objetivo en contexto	Transcurre la batalla	
Entradas		
Precondiciones	La batalla se inicio correctamente	
Salidas		
Poscondición si éxito	Se realiza la batalla.	
Poscondición si fallo	Se procede a reiniciar la batalla.	
Actores	Sistema	
Secuencia Normal	**Paso**	**Acción**
	1	Se consulta número de ejércitos atacante y se lanzan dados rojos. Si error S-1
	2	Se consulta número de batallones defensores y se lanzan dados azules. Si error S-2
	3	Se comparan dados.
	4	Se restan los batallones perdidos en la batalla.
	5	Si el enemigo no se ha quedado sin ejércitos y el atacante tiene mas de un ejercito en el territorio desde el cual ataca vuelve a 2
	6	Se actualiza tablero
Secuencias Alternativas	**Paso**	**Acción**
	s-1	Se lanzan los dados rojos de nuevo.
	s-2	Se lanzan los dados azules de nuevo.

Caso 5	Canjear cartas risk	
Objetivo en contexto	Conseguir batallones mediante el canjeo de cartas en la etapa de refuerzo	
Entradas		
Precondiciones	Sea el turno del jugador y este en la etapa de refuerzo	
Salidas		
Poscondición si éxito	Consigue nuevos batallones	
Poscondición si fallo	No consigue nuevos batallones, intentamos de nuevo.	
Actores	Jugador/Sistema	
Secuencia Normal	**Paso**	**Acción**
	1	Se seleccionan las tres cartas que queremos canjear. Si error S-1
	2	El sistema comprueba las cartas elegidas. Si error S-1
	3	El sistema suma el numero de batallones adicionales por el canjeo de cartas
Secuencias Alternativas	**Paso**	**Acción**
	S-1	No tenemos suficientes cartas.
	S-2	La secuencia escogida no es valida. Vuelve a 1.

Caso 6	**Mostrar carta objetivo**	
Objetivo en contexto	Visualizar el objetivo del jugador en la partida	
Entradas		
Precondiciones	Sea el turno del jugador	
Salidas		
Poscondición si éxito	Visualiza su objetivo	
Poscondición si fallo	No visualiza su objetivo	
Actores	**Jugador/Sistema**	
Secuencia Normal	**Paso**	**Acción**
	1	El usuario solicita ver el objetivo.
	2	El sistema muestra objetivo. Si error s-1
Secuencias Alternativas	**Paso**	**Acción**
	S-1	No se puede visualizar objetivo, se inicia la secuencia otra vez.

Caso 7	**Mostrar cartas risk**	
Objetivo en contexto	Visualizar las cartas risk que posee el jugador en ese momento de la partida	
Entradas		
Precondiciones	Sea el turno del jugador	
Salidas		
Poscondición si éxito	Visualiza sus cartas risk	
Poscondición si fallo	No visualiza sus cartas risk	
Actores	**Jugador/Sistema**	
Secuencia Normal	**Paso**	**Acción**
	1	El jugador solicita ver las cartas risk.
	2	El sistema visualiza las cartas risk. Si error s-1
Secuencias Alternativas	**Paso**	**Acción**
	S-1	No se puede visualizar las cartas risk, se inicia la secuencia otra vez.

Caso 8	**Defensa**	
Objetivo en contexto	**El jugador humano defiende su territorio**	
Entradas		
Precondiciones	Ataque dirigido por un jugador maquina	
Salidas		
Poscondición si éxito	Se realiza la batalla	
Poscondición si fallo	Reinicio de la aplicación	
Actores	**Jugador/Sistema**	
Secuencia Normal	**Paso**	**Acción**
	1	Se indica que has sido atacado.
	2	Se indica numero de batallones con el cual defiendes. (1 ó 2)
	3	El sistema realiza la batalla. S-1

Secuencias Alternativas	Paso	Acción
	S-1	La batalla no se realizó correctamente, se procede al reinicio de la aplicación.

Caso 9	Pasar de maniobra	
Objetivo en contexto	**Estando en la fase de ataque no hacer la fase de maniobra**	
Entradas		
Precondiciones	Sea el turno del jugador y este en la etapa de ataque	
Salidas		
Poscondición si éxito	Termina el turno	
Poscondición si fallo		
Actores	**Jugador/Sistema**	
Secuencia Normal	**Paso**	**Acción**
	1	El jugador activo deja pasar turno.
	2	El sistema le pasa el turno al siguiente jugador. Si error S-1
Secuencias Alternativas	**Paso**	**Acción**
	S-1	El jugador aún no ha hecho la fase de refuerzo

Caso 10	Pasar de ataque y maniobra	
Objetivo en contexto	**Estando en la fase de ataque no hacer la fase de ataque y maniobra**	
Entradas		
Precondiciones	Sea el turno del jugador y ha terminado la fase de refuerzo	
Salidas		
Poscondición si éxito	Termina el turno	
Poscondición si fallo		
Actores	**Jugador/Sistema**	
Secuencia Normal	**Paso**	**Acción**
	1	El jugador activo deja pasar el turno. Si error S-2
	2	El sistema le pasa el turno al siguiente jugador. Si error S-1
Secuencias Alternativas	**Paso**	**Acción**
	S-1	El jugador aún no ha hecho la fase de refuerzo

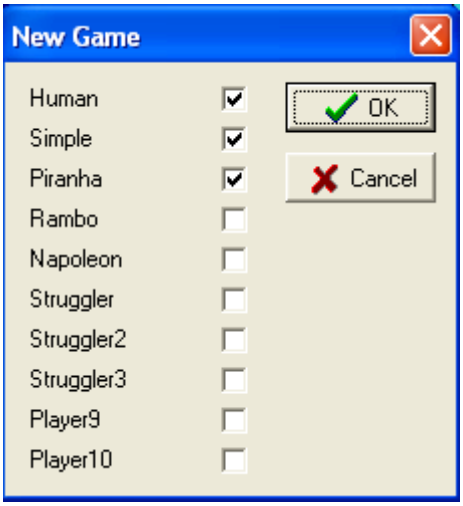
4.2. Prototipos de Interfaz de Usuario

A continuación especificamos las posibles secuencias de acciones que el sistema puede llevar a cabo con nuestra aplicación. Para comprender y especificar las interacciones entre actores humanos y el sistema hemos usado los siguientes prototipos de interfaz de usuario.

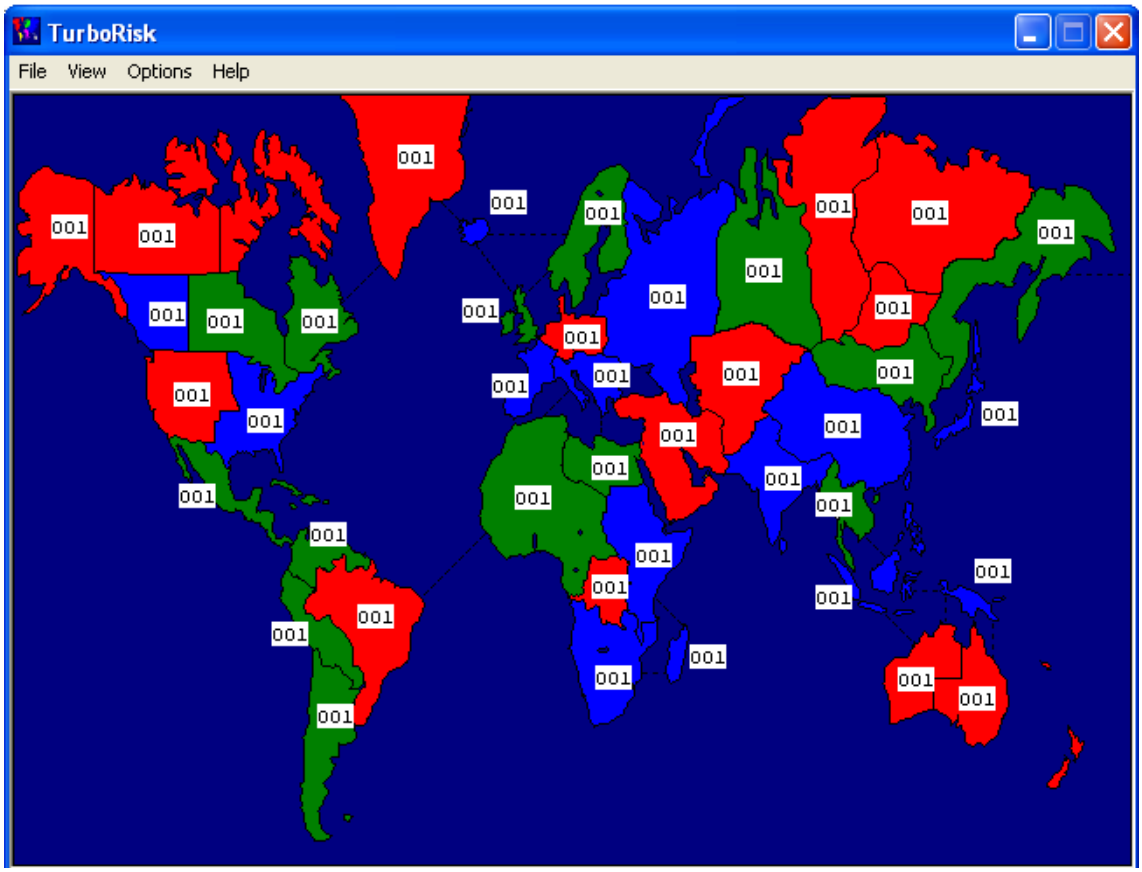
La secuencia que vamos explicar a continuación es la siguiente: el sistema comenzará la partida dando al jugador humano que esté delante del ordenador la posibilidad de elegir qué número de jugadores formarán el juego. Después se hará el reparto de los objetivos. Se mostrará al usuario el objetivo. Luego los jugadores humanos harán el reparto inicial de batallones y comenzará la batalla hasta que uno de los jugadores gane. Cuando ocurra esto el sistema lo anunciará y detendrá la partida.

Partiendo de los casos de uso hemos hecho unos modelos de interfaz gráfica y esquemas de pantalla, mediante composiciones de pantalla, algunas capturas de juegos, como pueden ser Turbo Risk. Con esto, hemos obtenido una secuencia de capturas (*storyboard)* para poder comprender bien el comportamiento de la aplicación.

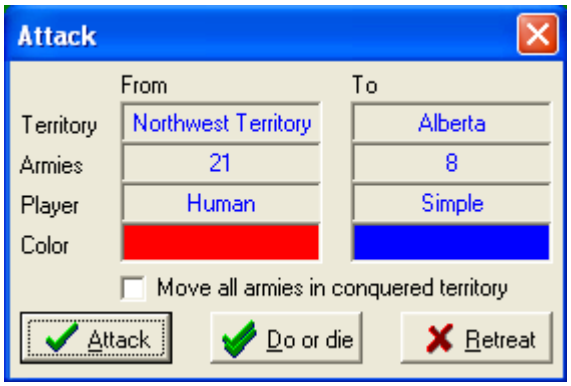
En primer lugar veremos la pantalla inicial donde se decide el número de jugadores y el tipo de los mismos.



En segundo lugar mostramos el tablero donde tendrá lugar todo el desarrollo del juego. A cada jugador se le asignará un color distinto y se colorearán todos los territorios que posea en ese mismo color. También se indicará el número de batallones que defienden el territorio.



En tercer lugar mostramos el posible formato que representará un ataque durante el juego. Una vez seleccionados los países del combate se mostrará una ventana que contendrá la información necesaria para el ataque. Esta es: el nombre y el número de batallones de cada territorio, los dos jugadores implicados en la batalla y controles para atacar o abandonar la batalla por parte del jugador atacante.



Para la fase de maniobra mostraremos una ventana igual que para la fase de ataque, pero ahora los territorios seleccionados pertenecen al mismo jugador y se le ofrece la posibilidad de movimiento de batallones entre los territorios siempre dejando uno como mínimo en cada territorio.

5. Análisis de Riesgos

A continuación se detalla nuestro primer análisis de riesgos así como la idea que teníamos de su evaluación, consecuencias y carácter. Mostramos también el seguimiento que hemos hecho de los mismos y los problemas que han surgido.

Añadimos una nueva columna a los riesgos para evaluar el impacto que han tenido en el desarrollo del proyecto. Los valores de esta columna serán: nulo, leve, moderado y alto.

TIPOS:

HUMANOS

	POSIBILIDAD	*CONSECUENCIA*	*IMPACTO EN EL PROYECTO*
Conflictos internos	Moderada	Catastrófica	Leve
Ausencia habitual	Alta	Seria	Moderado
Ausencia temporal	Moderada	Tolerable	Leve
Falta de motivación	Baja	Seria	Leve
Enfermedades	Baja	Seria	Nulo
Baja de un miembro	Muy baja	Seria	Nulo

TÉCNICOS:

	POSIBILIDAD	*CONSECUENCIA*	*IMPACTO EN EL PROYECTO*
Desconocimiento de tecnologías	Muy alta	Catastrófica	Moderado
Desconocimiento del área	Alta	Seria	Leve
Limitaciones del laboratorio	Alta	Seria	Alto
Falta de realismo	Alta	Seria	Leve
Perdida de datos	Moderada	Catastrófica	Nulo

Lagunas en la especificación	Baja	Seria	Leve
Monitorización en los laboratorios	Muy alta	Tolerable	Nulo
Control de las versiones	Moderada	Seria	Moderado

PLANIFICACIÓN:

	POSIBILIDAD	*CONSECUENCIA*	*IMPACTO EN EL PROYECTO*
Escasez de tiempo	Muy alta	Catastrófica	Moderado
Reparto de tareas	Muy alta	Tolerable	Moderado
Bajo objetivo	Baja	Seria	Nulo
Alto objetivo	Alta	Insignificante	Nulo

APLICACIÓN:

	POSIBILIDAD	*CONSECUENCIA*	*IMPACTO EN EL PROYECTO*
Refresco de pantalla	Moderada	Seria	Nulo
Jugabilidad	Muy alta	Tolerable	Leve
Entretenimiento	Baja	Tolerable	Leve
Gráficos excesivamente buenos	Muy baja	Tolerable	Leve

IMPREDECIBLES:

	POSIBILIDAD	*CONSECUENCIA*	*IMPACTO EN EL PROYECTO*
Huelgas	Muy baja	Seria	Nulo
Cierre de facultad	Muy baja	Seria	Nulo
Epidemias, catástrofes, guerras....	Muy baja	Catastrófica	Nulo

ANÁLISIS, PREVENCIÓN, SEGUIMIENTO Y RESULTADO

a) Humanos

Conflictos internos

Posibilidad: moderada **Consecuencia:** catastrófica
Métodos de prevención: toma de decisiones de la forma más democrática y ecuánime posible, tratando de solventar diferencias desde el diálogo y nunca desde la autoridad. Ser siempre flexible respecto a ideas ajenas.
Minimización de consecuencias: intentar mediar entre las partes en conflicto, tratando de llegar a una solución de compromiso que no perjudique a ninguna de las dos. En caso de diferencia insalvable encomendar a cada contendiente partes lo más independientes del proyecto y evitar su interrelación.
Seguimiento: no ha habido conflictos internos de forma abierta luego no han tenido impacto en el proyecto.
Resultado: no ha tenido consecuencias en el proyecto.

Ausencia habitual

Posibilidad: alta **Consecuencia:** seria
Métodos de prevención: identificar lo antes posible a las personas que no van a seguir el proyecto para excluirlas pronto de las tareas esenciales y evitar que paralicen el curso natural. Motivar y hacer entender a todos los miembros del grupo que la presencia de todos es fundamental para el correcto desarrollo del proyecto.
Minimización de consecuencias: reestructurar la planificación y el reparto de tareas considerando estas ausencias. Si así el objetivo inicial es inalcanzable, tratar de eliminar del mismo las partes necesarias para que sea accesible al nuevo grupo de trabajo.
Seguimiento: la ausencia habitual ha sido un gran problema en el proyecto. La realización y uso de una lista de asistencia y de trabajo realizado supuso una fuerte mitigación de este riesgo. Sin embargo, nunca hemos logrado reducir a cero este problema.
Resultado: este problema ha causado desánimo en miembros del grupo y retraso o falta de cumplimiento de algún objetivo en las distintas iteraciones.

b) Técnicos

Desconocimiento de la tecnología

Posibilidad: muy alta **Consecuencia:** catastrófica
Métodos de prevención: repartir las tareas de investigación de cada tecnología a cada miembro del grupo y realizar sesiones de intercambio de conocimientos. Fijarse plazos suficientes para aprender lo suficiente de cada tecnología y realizar pruebas sencillas. Centralizar la documentación relacionada a las tecnologías para facilitar la búsqueda.
Minimización de consecuencias: división de la investigación/implementación entre las personas con mejores condiciones (de tiempo, de experiencia previa, etc.) dedicándose los demás a las demás tareas. Realizar sesiones de búsqueda conjunta y aprendizaje intensivo.

Seguimiento: hemos tenido problemas con la implantación de la red neuronal en el proyecto.
Resultado: A través de un detallado estudio de las tecnologías mediante diversos métodos se consigue un dominio total de las mismas.

Desconocimiento del área

Posibilidad: alta **Consecuencia:** seria
Métodos de prevención: búsqueda e intercambio de información relativa al área del proyecto.
Minimización de consecuencias: fijarse plazos y niveles de conocimiento mínimos antes de continuar con el proyecto. Realizar sesiones intensivas de búsqueda e intercambio de información. En última instancia aplicar soluciones imaginativas en aquellos aspectos desconocidos.
Seguimiento: No fue necesario porque en ningún momento representó peligro alguno.
Resultado: gracias al conocimiento adquirido en las búsquedas de información y un dominio previo del área por parte de los integrantes del grupo el resultado fue satisfactorio.

Limitaciones del laboratorio

Posibilidad: alta **Consecuencia:** seria
Métodos de prevención: preguntar y descubrir las limitaciones de los laboratorios. Preguntar a técnicos sobre los aspectos técnicos que más problemas puedan dar. Realizar pequeñas pruebas de las tecnologías necesarias en los laboratorios.
Minimización de consecuencias: tratar de efectuar los menos cambios posibles para conseguir utilizar la tecnología elegida aunque su rendimiento/implementación no sean las mejores. Buscar sistemas alternativos donde ejecutar el proyecto a sus partes conflictivas. En última instancia, migrar a una tecnología que se sabe a ciencia cierta disponible en los laboratorios.
Seguimiento: además de la falta de puestos libres, hemos tenido problemas con el entorno de programación Borland C++Builder instalado además sólo en 6 laboratorios. Se nos han producido bloqueos del programa sistemáticos por lo que cada línea de código que escribíamos había que guardarla (y justo cuando guardabas era cuando más probabilidad había de bloquearse). Hemos ido avisando a los técnicos a lo largo del curso pero no se ha encontrado solución al tema. Esto ha afectado al desarrollo normal y al ánimo del personal, ya que hay días que han sido desesperantes.
Resultado: debido al problema anterior, hemos tenido que reescribir fragmentos de código varias veces en un mismo día, lo que nos ha producido una gran pérdida de tiempo y falta de productividad.

Falta de realismo

Posibilidad: alta **Consecuencia:** seria
Métodos de prevención: informarse y documentarse sobre el correcto funcionamiento y reglas del juego risk.
Minimización de consecuencias: comenzar con una funcionalidad básica e ir ampliándola según importancia y uso en la vida real.
Seguimiento: se ha intentado hacer la aplicación lo más real posible, teniendo que ir mejorando el diseño poco a poco, por no satisfacer una calidad en el realismo.
Resultado: hemos conseguido una aplicación que se acerca bastante a la realidad, sabiendo que hay aspectos que no hemos implementado, aunque si considerados, que serían necesarios para conseguir un realismo total.

Pérdida de datos

Posibilidad: moderada **Consecuencia:** catastrófica
Métodos de prevención: garantizar que cada modificación efectuada en el proyecto sea guardada por cada miembro del subgrupo implicado, y las generales por todos los miembros.
Minimización de consecuencias: almacenamiento por parte del jefe de la última versión y permitir la lectura a los miembros del grupo. Además permitir la comunicación vía correo electrónico entre los miembros del grupo y el jefe. Almacenamiento de seguridad en el servidor ftp de las configuraciones del proyecto finalizadas y temporales de las que estén en desarrollo, así como las versiones de prueba que se generen.
Seguimiento: no hemos tenido ningún problema de pérdida de datos, gracias a nuestra gestión de configuración, en la cual guardamos todas las versiones en un ftp, además de enviárnoslas por correo para tener otra copia más.
Resultado: no hemos tenido problemas por pérdida de datos, a excepción de los comentados en “Limitaciones de laboratorio” por problemas de bloqueos inesperados del C++Builder en distintos ordenadores y distintos laboratorios.

Control de las versiones

Posibilidad: moderada **Consecuencia:** seria
Métodos de prevención: identificar con el número de versión y la fecha realización de la última actualización
Minimización de consecuencias: no borrar las versiones anteriores ni sobrescribir una versión previa.
Seguimiento: dado que se ha trabajado de manera concurrente sobre muchas clases, nuestro mejor método de control de versiones ha sido identificar con la fecha sobre todo las integraciones parciales y con comentarios acerca de si falta algún paquete por integrar o no.
Resultado: Con una numeración con fecha y anotaciones en un fichero de texto de los cambios o mejoras realizadas, nos ha sido suficiente para gestionar las versiones de trabajo y de paquetes, por lo que no nos ha resultado problemático.

c) Planificación

Escasez de tiempo

Posibilidad: muy alta **Consecuencia:** catastrófica
Métodos de prevención: fijar plazos no excesivamente dilatados ni concentrados pero ser bastante firmes en su consecución. Considerar de manera realista la envergadura de cada tarea y las circunstancias de cada grupo de cara a la planificación y gestión. Permitir una planificación/gestión flexible.
Minimización de consecuencias: reestructuración de las tareas para tratar de aumentar la productividad de cada grupo. Reestructuración de la planificación intentando ser lo más realista posible. Establecer penalizaciones o gratificaciones para quien cumpla plazos.
Seguimiento: se ha tenido la necesidad de posponer objetivos de iteración en iteración, de todas formas tras la primera iteración nos dimos cuenta de que el alcance había que recortarlo, e ir reestructurando nuestro plan de fase para garantizar que el funcionamiento de lo que se hacia era satisfactorio y cumplía la gestión de calidad.
Resultado: algunos objetivos previstos para la última iteración nos han podido cumplirse, quedando partes del proyecto sin terminar, como la construcción de una interfaz más sofisticada.

Reparto de tareas

Posibilidad: muy alta **Consecuencia:** tolerable
Métodos de prevención: adjudicar a cada miembro del grupo aquellas tareas en las que se encuentre más cómodo y capacitado para su realización.
Minimización de consecuencias: No sobrecargar a ningún miembro de grupo con tareas muy pesadas, costosas y esenciales para evitar riesgos humanos.
Seguimiento: cumplimos los métodos de prevención y minimización de consecuencias prácticamente, pero el problema ha estado en que no se ha cumplido al 100%.
Resultado: hemos sufrido retrasos en los plazos marcados debido a que una tarea muy concreta la ha realizado una sola persona. El retraso de esta tarea produjo que otras se viesen paradas hasta que se terminó de realizar la primera.

d) Aplicación

Refresco de pantalla

Posibilidad: moderada **Consecuencia:** seria
Métodos de prevención: asegurarse de que el sistema transmite las órdenes en el tiempo preciso y actualiza el tablero en un tiempo admisible.
Minimización de consecuencias: no sobrecargar al sistema con información no necesaria y tener una política de prioridad al refresco frente a otras peticiones.
Seguimiento: En ningún momento el sistema se vio afectado por sobrecarga de información por lo que siempre fue capaz de mantener una política de refresco adecuada.
Resultado: no hemos tenido ningún problema con el refresco del sistema, es más, puede aumentarse más, aunque hay que tener en cuenta la limitación de los ordenadores de los laboratorios.

6. Estimación

Hemos realizado un cálculo de estimación para calcular una medida aproximada de tiempo, esfuerzo y coste. No hemos entrado un análisis muy exhaustivo de productividades individuales pues tenemos pocas estadísticas de cada miembro del grupo. Hemos realizado la estimación basándonos en casos de uso porque es la manera más concreta que tenemos a nuestro alcance para poder hacer una estimación posterior en puntos de función y en líneas de código, pues directamente de los requisitos no tenemos claridad de ello. Analizamos por tanto los casos de uso y los prototipos de interfaz para conocer el número medio de entradas, salidas, consultas y ficheros para obtener estimaciones.

INICIO DE APLICACIÓN

Entradas:

- Número jugadores humanos
- Número jugadores máquina
- Botón OK.
- Botón CERRAR APLICACIÓN
- Tipo inteligencia 1
- Tipo inteligencia 2

Salidas:

- Validación número jugadores máquina
- Validación número jugadores humanos.

Consultas:

- Ayuda

FASE ATAQUE

Entradas:

- País atacante
- País defensor.
- Botón ataque.
- Salir
- Número de batallones que atacan.
- Número de batallones que defienden.
- Número de batallones traspasados al territorio conquistado.

Salidas:

- Nuevo número batallones en territorio atacante
- Nuevo número batallones en territorio defensor.
- Nuevo propietario territorio conquistado.

Consultas

- Cartas risk
- Objetivo
- Ayuda

FASE MANIOBRA

Entradas:

- País emisor
- País receptor
- Botón maniobra
- Número de batallones en territorio emisor.
- Número de batallones en territorio receptor.
- Cancelar

Salidas:

- Nuevo número batallones en territorio emisor
- Nuevo número batallones en territorio receptor.

Consultas

- Cartas risk
- Objetivo
- Ayuda

PANTALLA PRINCIPAL

Entradas:

- Nuevo juego
- Salir
- País Atacante
- País Defensor
- País Emisor
- País Receptor

Salidas:

- Panel Información
- Zoom in en territorios
- Zoom out en territorios

Consultas

- Cartas risk
- Objetivo
- Ayuda

ESTIMACIÓN BASADA EN PUNTOS DE FUNCIÓN (FP)

Valores Estimados PF			
	Simple	Media	Compleja
Entradas del usuario	15	25	45
Salidas del usuario	9	15	30
Consultas del usuario	7	15	25
Ficheros	1	4	8
Interfaces externos	4	4	4

Pesos de complejidad				
	Simple	Media	Compleja	**Elegida**
Entradas del usuario	3	4	6	**4**
Salidas del usuario	4	5	7	**5**
Consultas del usuario	3	4	6	**4**
Ficheros	7	10	15	**10**
Interfaces externos	5	7	10	**7**

Pesos Complejidad media			
	Valor Estimado	Peso Asignado (P)	Cuenta por característica (C)
Entradas del usuario	25	4	100
Salidas del usuario	15	5	75
Consultas del usuario	15	4	60
Ficheros	4	10	40
Interfaces externos	4	7	28

Corrección de Complejidad

Valor total = ∑ P * C = 303 (Usamos valores estimados simples asignando pesos medios de complejidad, descartando posibles ampliaciones)

Donde
P = peso asignado para la característica
C = cuenta de la característica

Suma Respuestas (Corrección de Complejidad) = 44

Factor de ajuste = 0,65 + 0,01 * 48 = 1.13

FP = Valor total * Factor de ajuste
= 303 * 1.13 = **342**

- El esfuerzo total que habrá que dedicar sería:
  Esfuerzo total = Número estimado de FP / Media de productividad
  = 342 / 6,5 = **52 personas mes**

(Estimamos productividad en 6,5 FP/persona mes)

ESTIMACIÓN BASADA EN LÍNEAS DE CÓDIGO (LOC)

Calculamos las LOCs a partir de FPs usando el número medio de LOC por FP (media) para un lenguaje dado (AVC), en nuestro caso C++.

–LOC = AVC * número de puntos de función
–AVC para lenguaje C++ = 36

LOC = 36 * 303 = **10908 líneas de código**

Estimando la productividad en unas 300 LOC / P-mes

- Esfuerzo total = 10908 / 300 = **36 personas mes**

COMPARACIÓN DE ESTIMACIONES

- FP
  56 personas mes
- LOC
  36 personas mes

COSTE

3 personas * 9 meses 160 horas al mes (8 horas al día) a 10€ la hora de programador supone un coste de:

43.200 €uros

7. Tarjetas CRC

7.1. Brainstorm

A continuación mostramos una tabla con los términos recogidos. Procedimos a ello en grupo por turnos democráticamente. Posteriormente fuimos filtrando las propuestas que no fuesen relevantes y nos quedamos con las más relevantes, usamos para las tarjetas CRC. Este es el primer "*brainstorm*" que realizamos:

VENTANA	JUEGO
ATAQUE	ESTRATEGIA
TERRITORIO	REFUERZO
SOLDADO	MUNDO
MAPA	PARTIDA
JUGADOR	SIMULACIÓN
INTELIGENCIA ARTIFICIAL	VISTA 3D
C++	AZAR
MANIOBRA	DADOS
BATALLA	CONTINENTE
PAÍS	CABALLERÍA
ESTADÍSTICA	CONQUISTA
ARCHIVO	DESTRUCCIÓN
SALIDAS	TABLERO
ÉXITOS	FALLOS
DERROTA	ASTUCIA
VICTORIA	SONIDOS
GUERRA	ANIMACIÓN
OBJETIVO	RED NEURONAL
INTERFAZ APLICACIÓN	CLIENTE
RED	PUNTOS DE CONTROL
PAQUETE	MENSAJE
REFRESCO	EVENTO
TREGUA	COMPETICIÓN
ACUERDOS	VENGANZA
TRATADOS	PROPIETARIO
DEFENSA	COLOR
BATALLÓN	FICHA
TANQUE	CARTAS
ENEMIGO	INFANTERIA
REGLAS	TURNO
ARTILLERÍA	ALIANZA
HUMANO	SISTEMA EXPERTO
MAQUINA	MERCANCÍA
ALGORITMOS EVOLUTIVOS	CONSOLA
TRAMPA	CANJEAR
OCEÁNO	ALGORITMOS BÚSQUEDA
FRONTERA	ESPERANZA

7.2. Tarjetas CRC

Nos facilitan la estimación del conjunto de clases obtenida hasta este punto respondiendo bien a las necesidades dinámicas de la aplicación. Nos suponen una primera aproximación a los objetos que luego se van a utilizar, y han sido creadas a partir de la lista de clases básicas. Una vez obtenidas éstas, nos dividimos en grupo para su elaboración:

CLASE	***COLABORADORES***
Punto	
RESPONSABILIDADES	
- Establecer una coordenada	

CLASE	***COLABORADORES***
Mapa	-Territorio -Continente
RESPONSABILIDADES	
-Gestiona la información en el tablero a través de un grafo.	

CLASE	***COLABORADORES***
Continente	
RESPONSABILIDADES	
-Agrupación de una serie de territorios	

CLASE	***COLABORADORES***
Objetivo	
RESPONSABILIDADES	
-	

CLASE	***COLABORADORES***
Objetivo (clase padre)	
SUBCLASES	
-Objetivo24territorios -ObjetivoDestruirEnemigo -Objetivo2Continentes -Objetivo18Territorios	
RESPONSABILIDADES	
- Establecer los distintos tipos de objetivos	

CLASE	***COLABORADORES***
Objetivo24Territorios	
SUPERCLASE	
Objetivo	
RESPONSABILIDADES	
-El jugador tiene que conquistar 24 territorios y mantenerlos para cumplirlo.	

CLASE	***COLABORADORES***
Objetivo18Territorios	
SUPERCLASE	
Objetivo	
RESPONSABILIDADES	
-El jugador tiene que conquistar 18 territorios, teniendo como minimo dos batallones en cada uno y poseerlos a la vez para cumplirlo.	

CLASE	***COLABORADORES***
ObjetivoDestruirEnemigo	
SUPERCLASE	
Objetivo	
RESPONSABILIDADES	
-El jugador tiene que destruir a un enemigo determinado para cumplirlo.	

CLASE	***COLABORADORES***
Objetivo2Continentes	
SUPERCLASE	
Objetivo	
RESPONSABILIDADES	
-El jugador tiene que conquistar dos continentes determinados para cumplirlo.	

CLASE	***COLABORADORES***
Ventana Interfaz GND	-Vista 3D -Superclase Interfaz
SUPERCLASE	
Ventana Interfaz	
RESPONSABILIDADES	
-Mostrar toda la parte grafica de la aplicación para usuario rodadura	

CLASE	***COLABORADORES***
Juego	-Mapa -Objetivo -Jugador
RESPONSABILIDADES	
-Lleva a cabo la ejecución de la partida	

CLASE	***COLABORADORES***
-Jugador	-Situación -Objetivo -Territorio
SUBCLASE	
-Jugador Humano -Jugador Máquina	
RESPONSABILIDADES	
-Realiza las acciones de cada jugador individualmente.	

CLASE	***COLABORADORES***
Jugador Humano	-Situación -Objetivo -Territorio
SUPERCLASE	
Jugador	
RESPONSABILIDADES	
-Se encarga de realizar todas las acciones que el usuario le indica como jugador.	

CLASE	***COLABORADORES***
Jugador Máquina	-Situación -Objetivo -Territorio
SUPERCLASE	
Jugador	
RESPONSABILIDADES	
-Se encarga de realizar todas las acciones que durante el turno de un jugador controlado por el ordenador	

CLASE	***COLABORADORES***
Inteligencia	
SUBCLASES	
-Inteligencia Sistema Experto -Inteligencia Red Neuronal	
RESPONSABILIDADES	
-Se encarga de razonar las acciones del jugador máquina.	

CLASE	***COLABORADORES***
Inteligencia Sistema Experto	
RESPONSABILIDADES	
-Se encarga de razonar las acciones del jugador máquina a través de un sistema experto	

CLASE	***COLABORADORES***
Inteligencia Red Neuronal	
RESPONSABILIDADES	
-Se encarga de razonar las acciones del jugador máquina a través de una red neuronal	

CLASE	***COLABORADORES***
VentanaAtaque	-Territorio
RESPONSABILIDADES	
-Visualizar el combate entre dos territorios	

CLASE	***COLABORADORES***
VentanaManiobra	-Territorio
RESPONSABILIDADES	
-Visualiza el movimiento de batallones entre dos territorios colindantes.	

CLASE	***COLABORADORES***
VentanaTablero	-Mapa
RESPONSABILIDADES	
-Visualiza el tablero.	

CLASE	***COLABORADORES***
VentanaInicioPartida	-Jugador
RESPONSABILIDADES	
-Gestiona los parámetros iniciales de la partida: numero de jugadores y tipo de los mismos.	

8. Gestión de Configuración

8.1. Estándares

Disponemos de estándares de código y distintas plantillas de documentación, con lo que se facilita la legibilidad y garantizamos una calidad y homogeneidad en los documentos. Estos archivos se encuentran en archivos adjuntos.

8.2. Gestión de configuración

Nuestra gestión de configuración es algo rudimentaria, simple pero eficaz, si bien tiene el defecto de no servir para proyectos que incumban a un mayor número de personas, motivo que se explicará más adelante.

Para gestionar nuestro proyecto hemos dispuesto de un servidor ftp en el que se ha guardado tanto la documentación generada, ya sean informes de reuniones como plantillas de diseño, documentos de requisitos, etc. y en sus correspondientes carpetas el código que íbamos generando en cada iteración agrupado por paquetes. Con el fin de evitar la consecuencia del riesgo de pérdida de datos, la única persona que tenía permiso para escribir información en el ftp era el jefe de documentación. Esta restricción solo implicaba a la escritura, pues la lectura estaba autorizada para todos los miembros del grupo, gracias al uso de permisos en la gestión del servidor.

También hemos ido almacenando las integraciones realizadas cada cierto tiempo, así como las versiones estables, de prueba o entregas. Cada vez que se iniciaba una nueva iteración, se partía de la última versión estable de la anterior iteración, por lo que de esta forma hemos asegurado tener siempre, en todo momento, una versión estable del sistema.

8.3. Gestión de cambios

- La gestión de cambios se realiza de forma ordenada. Una vez estructuradas las acciones que llevarían a cabo, se repartían equitativamente entre los tres miembros del grupo.
- Cada semana los componentes del grupo envían al jefe de documentación los paquetes actualizados y es él el que se encarga de actualizar el proyecto general y de subirlo al servidor FTP. Sólo el jefe de documentación tiene permiso de escritura y el resto del grupo tiene permiso de lectura. De esta forma minimizamos riesgos de pérdida de información y de actualización errónea de versiones.

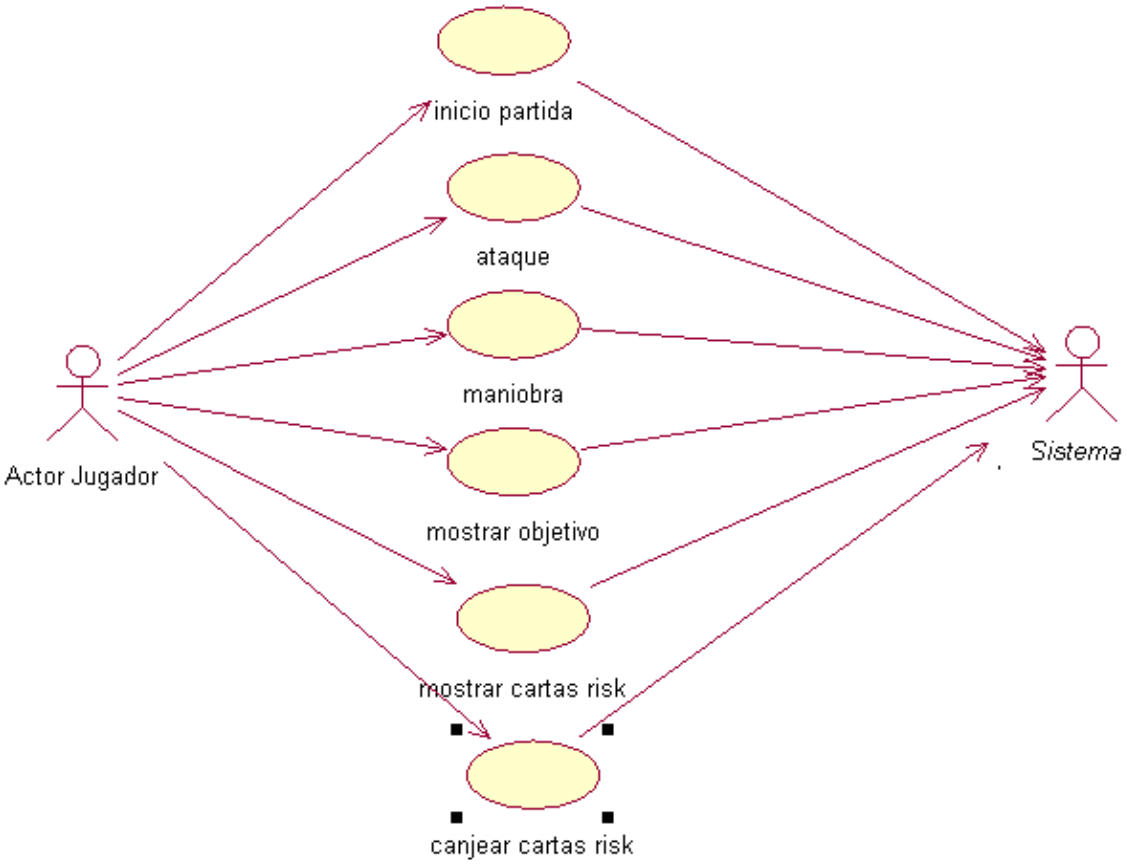
9. Diagramas UML

UML (Unified Modeling Language) tiene el propósito de realizar un modelado de los requisitos del sistema, con la finalidad de una mayor expresión en la representación de diversos diagramas que explican detalladamente la funcionalidad del sistema partiendo de los casos de uso.

El estándar de OMG define nueve tipos de diagramas. De todos estos, hemos escogido los más representativos para nuestro diseño y tenemos:

- Funcionales: Muestran la funcionalidad del sistema desde el punto de vista del usuario:
    - Diagramas de caso de uso
- Objetos: Muestran la estructura y la subestructura del sistema usando objetos, atributos, operaciones y asociaciones:
    - Diagramas de clase
- Dinámicos: Muestran el comportamiento del sistema:
    - Diagramas de secuencia

A continuación se encuentran el diagrama de caso de uso que nos da una visión de la funcionalidad del sistema.


inicio partida
ataque
maniobra
mostrar objetivo
mostrar cartas risk
canjear cartas risk
Actor Jugador
Sistema

10. Patrones de diseño

En primer lugar decir que no hemos aplicado patrones de forma estricta. Sin embargo, si realizamos aproximaciones a los mismos. Los patrones que pueden ser diferenciados en la aplicación son los siguientes:

Observer: tiene como objetivo definir una dependencia uno a muchos entre objetos de tal forma que cuando un objeto cambie de estado, todos los objetos que dependan de él sean notificados y actualizados automáticamente.

En nuestra aplicación, el sistema actualiza el tablero constantemente en función de los resultados de los ataques, actualiza propietario y numero de batallones.

Mediator: el mediador coordina las interacciones entre sus asociados

El sistema coordina las acciones entre los distintos jugadores que forman parte de la partida.
Command: encapsula una petición como un objeto, permitiendo parametrizar clientes con diferentes peticiones y soportar acciones que se puedan deshacer.

Este patrón queda reflejado en el caso de que se añadiera la posibilidad de jugar en red.

Puente: desacopla una abstracción de su implementación de manera que las dos puedan evolucionar independientemente.

En nuestro proyecto lo aplicamos a todas los tipos de inteligencia artificial.

11. Implementación

11.1. Introducción

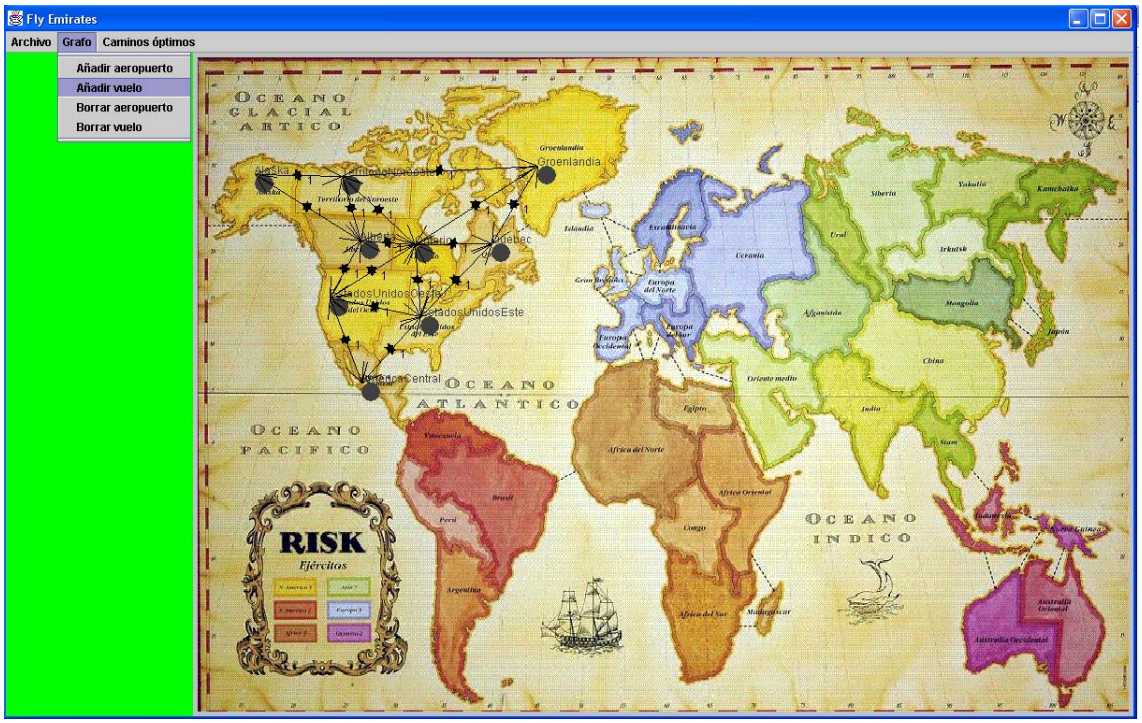
Una vez que hemos desarrollado un análisis y un diseño completo y adecuado para nuestro sistema, el siguiente paso que hay que efectuar en un proyecto es el de implementar.

En nuestro proyecto existen dos bloques generales e independientes que engloban casi toda la funcionalidad del proyecto. El primer bloque es el que corresponde a la parte gráfica o visual del sistema, es decir, lo que el usuario ve del sistema. El segundo bloque es aquel que denominamos núcleo de aplicación. Este es el que hace posible el funcionamiento de la partida y la inteligencia de los jugadores máquina. El primer bloque mencionado antes visualiza el segundo bloque.

11.2. Núcleo de la aplicación

Puede decirse que el núcleo es el cerebro de la aplicación. El objetivo de la capa núcleo es poder crear un funcionamiento de la aplicación internamente.

La clase principal es Juego. En la constructora de esta clase leemos de fichero la información sobre los territorios, sobre los continentes y sobre los objetivos. Con la información sobre los territorios creamos la clase Mapa, que es la que contiene la lista de los territorios, así como un grafo de conexiones y de caminos mínimos entre ellos implementado mediante el algoritmo de Floyd, indispensable para el desarrollo del juego. La información sobre los territorios, a saber, nombre, coordenada (útil para la parte gráfica), continente al que pertenece, etc., y la información sobre las conexiones entre los territorios, la introdujimos usando una práctica que hizo uno de nosotros en LP3, consistente en un editor de grafos, con aeropuertos y vuelos entre los aeropuertos. Se cargó de fondo la foto del tablero para saber el punto exacto donde colocar cada aeropuerto y se introdujeron los 42 aeropuertos, uno por cada territorio. Después se introdujeron los vuelos entre cada par de territorios que estaban conectados en el mapa. Por ejemplo, una vez metidos todos los aeropuertos y los vuelos de Estados Unidos, el panel tenía el aspecto visible en la siguiente figura. Se puede observar el menú de introducir aeropuertos y vuelos, así como los aeropuertos ya metidos y las conexiones entre ellos. Los números en las conexiones son el coste de los vuelos y las flechas indican que son trayectos bidireccionales. Para nuestro proyecto suponemos que el coste de cada trayecto es 1.



Toda esta información se guardó en un fichero que es el que se lee al crear la clase Juego. Hay que decir que esa práctica estaba en Java, pero no influía sobre nuestro proyecto hecho en C++ ya que solo utilizaríamos el archivo de texto generado.

Mediante un menú de opciones, se puede elegir la configuración que tendrá la partida, a elegir entre las siguientes opciones: 2 jugadores Máquina (Sistema Experto vs Red Neuronal), 2 jugadores Máquina (los 2 con sistema experto), de 3 a 6 jugadores Máquina (todos con sistema experto), 1 humano y 1 jugador Máquina (pudiendo ser sistema experto o red neuronal) y 1 humano y entre 2 y 5 jugadores Máquina (todos ellos con sistema experto). Es decir, la red neuronal solo se puede usar cuando son 2 jugadores puesto que está orientada a conquistar el mundo. Estas opciones se eligen automáticamente, el usuario sólo tiene que elegir el número de jugadores, si habrá jugador humano y el tipo de inteligencia (sistema experto o red neuronal). Esta última opción sólo tendrá efecto si se eligen 2 jugadores.

También se puede elegir la opción de que te avisen cuando te atacan, para así ver los dados que saca el atacante y los dados que sacas tú. Si se elige esta opción, en el momento en que te atacan se abre una ventana de TformAtaque en la que se ven los dados y posteriormente a los 3 segundos se cierra sola y continúa la partida.

Dependiendo de las opciones elegidas para la partida, el método iniciaJuegoSistemaExperto crea un vector de Jugadores (puede ser un JugadorHumano y el resto JugadorMaquina o todos JugadorMaquina), reparte aleatoriamente los territorios entre los jugadores, reparte los objetivos: si son 2 jugadores el objetivo será conquistar el mundo, si son de 3 a 6 el objetivo será conquistar 2 continentes (ver posibles combinaciones) o conseguir 24 territorios. Una vez que se sabe el objetivo de cada jugadorMaquina se procede a crear su atributo Inteligencia, pudiendo ser de tipo red neuronal, aleatoria o sistema experto, que vemos detallado en el punto 11.2.1.

INTELIGENCIA SISTEMA EXPERTO

El sistema experto se basa en comprobar la situación de la partida y en función de ella aplicar una determinada regla, es decir:

Si *Condición* entonces *Acción*.

El sistema experto establece patrones (situación, acción) que indican cómo reaccionar ante una situación.

Estas reglas han sido pensadas minuciosamente desde la experiencia que tenemos de este juego y son las que cualquiera aplicaría en algún determinado momento, intentando abstraerlas lo máximo posible.

Ya que el turno de un jugador se divide en refuerzo, ataque y movimientos habrá reglas para cada fase. Aquí hay que hacer notar que las reglas están muy relacionadas entre sí, por ejemplo, no puedes atacar cierto objetivo si antes no refuerzas los territorios desde los que vas a atacar.

Antes de la fase de refuerzo hay que decidir si queremos canjear las cartas que tengamos, por lo que también habrá reglas para decidirlo.

Reglas del sistema experto:

1.Canjeo de cartas:

Lo primero que se hace aquí es comprobar si se puede canjear y si es así cuántos batallones extra obtendríamos, pudiendo ser 10, 7, 5 ó 3.

Hay que recordar que el número máximo de cartas que podemos guardar son 5, así que vamos sacando las condiciones:

-Si el canjeo son 10 o tenemos 5 cartas en la mano, la decisión es canjear.

-Si el canjeo es 3, 5 ó 7 se comprueba la situación en la partida, para obtener información sobre si va bien o mal el jugador, mirando el número de posesiones y también la proporción entre número total de batallones y número de posesiones para ver si está muy desperdigado o no. Se comparan estos números con unos valores mínimos asociados a si son 3, 5 ó 7 batallones el posible canjeo y si no se superan esos valores se decide canjear.

2.Refuerzo:

Regla: Estar a punto de ganar.

Condición: Esta regla se aplica cuando la probabilidad total de conseguir todos los objetivos que te faltan es alta.

Recordamos que la probabilidad normal con la que se decide atacar de un territorio a otro es que sea mayor o igual a 0.5. Si se mira en el código, se verá que aplicamos esta regla cuando la probabilidad total es mayor o igual que 0.3. Este valor puede parecer bajo pero hay que tener en cuenta que aún tiene q reforzar en esos territorios desde los q va a atacar y esa probabilidad realmente subiría mucho después del refuerzo. Además cuando hay una situación no favorable para ganar este valor suele ser 0.000... Esto es debido a que se calcula multiplicando las probabilidades parciales,

por ejemplo: 0.8 * 0.8 * 0.8 * 0.8 * 0.8= 0.32, y va bajando cada vez más con cada territorio q hay q conquistar.

El problema es que si desde el mismo territorio tenemos que atacar 2 territorios, la probabilidad de ganar cada uno de esos 2 territorios se calcula independientemente sin tener en cuenta que cuando ataque al segundo territorio ya no tendrá tantos batallones como al principio. Es por esto por lo que hay que llegar a un compromiso con el valor con el que hay que comparar la probabilidad total. Si ponemos un valor como 0.3 tendremos el problema anterior, si ponemos un valor alto como al principio (habíamos puesto 0.8) tendremos el problema de que si no tenemos que atacar 2 o más territorios desde el mismo, la probabilidad total no reflejará realmente la probabilidad que tenemos de ganar.

Por lo tanto, suponiendo que esta regla es la última que se llama antes de acabar la partida, si ponemos un valor bajo se aplicará seguramente un par de veces, cuando si ponemos un valor alto solo se aplicaría una vez o tal vez ninguna (si el último territorio tiene 1 batallón y yo tengo 3, la probabilidad no es mayor que 0.8, por lo que la regla no se aplicaría y por el contrario sí que estoy a punto de ganar).

Acción: En este caso se refuerzan los territorios colindantes a los objetivos, olvidándose de todo lo demás.

Regla: Ir a por tu objetivo.

En esta regla la acción es como la regla anterior solo que aquí no importa la probabilidad total en la condición.

Regla: Reforzar continentes amenazados.

Condición: esta regla se aplica si se tiene algún continente entero y hay alguna frontera que está amenazada. Por amenazada entendemos que tenga algún enemigo adyacente que tenga al menos el mismo número de batallones que la frontera. Por ejemplo, si tengo 5 batallones en una frontera y un enemigo tiene 5 también, el enemigo en su turno podría reforzar ese territorio y así obtener superioridad para atacar mi continente.

Acción: reforzar esas fronteras amenazadas, superando hasta en 3 batallones al enemigo siempre que podamos, es decir, que tengamos suficientes batallones para reforzar.

Regla: Reforzar contra continente enemigo.

Condición: que haya algún enemigo que tenga un continente entero, que tengamos un territorio adyacente a ese continente y que la probabilidad de ganar desde ese territorio supere una probabilidad mínima. Por ejemplo, si tengo 2 batallones y el enemigo tiene 15, lo mejor es no desperdiciar las tropas y reforzar en otro sitio.

Acción: se refuerza ese territorio con la probabilidad más alta de ganar desde el que atacar al continente. Hay que apuntar que no es lo mismo dejar que un enemigo cobre por Asia (7), a que cobre por Oceanía (2), así que esta probabilidad mínima depende del continente que queramos atacar, siendo más baja para Asia y más alta para Oceanía y América del Sur (2 batallones también).

Regla: Reforzar continente fácil.

Condición: que haya algún continente que estoy a punto de conquistar, comprobando que la proporción entre los batallones que tengo que vencer y los territorios que me faltan sea baja. Además de esto se comprueba que la proporción entre los territorios que tengo y los que me faltan sea cercana a 1.

Acción: se refuerza el territorio que más batallones tenga que vencer a su alrededor, la cantidad que se refuerza es igual al número de batallones enemigos totales en ese continente, intentando superar ese número en 3.

Hay casos en continentes grandes en los que esta acción podría no ser la deseada para conquistar un continente en un único turno si se pudiera, ya que se refuerza todo en el mismo territorio en vez de repartirlo. Pero esta opción es la más segura y a lo mejor en un turno no se consigue, pero así no se desperdiga demasiado y se queda muy fácil para los turnos siguientes.

3.Ataque:

Las reglas de ataque coinciden con las de refuerzo. Una vez que te has reforzado bien, las reglas de ataque podrían únicamente consistir en atacar donde tengas buena probabilidad de ganar, pero también hay que comprobar que donde atacas es donde más te interesa. Por eso las reglas de ataque siguen el patrón de las de refuerzo.

Hay que decir que las reglas de ataque se aplican una a una recursivamente, es decir, cada vez que realiza un ataque se vuelve a comprobar todo para realizar el siguiente ataque. Esto se hace así porque a lo mejor empiezas el turno atacando para conseguir un continente fácil, pero según vas atacando es posible que apliques la regla de estar a punto de ganar o la de ir a por un continente enemigo. Las reglas son:

Regla: Estar a punto de ganar.

Condición: es la misma que en la regla de refuerzo

Acción: se ataca desde el territorio que más probabilidad tenga de ganar a su objetivo.

Regla: Ir a por tu objetivo.

Condición: es la misma que en la regla de refuerzo

Acción: se ataca desde el territorio que más probabilidad tenga de ganar a su objetivo.

Regla: Ir a por un continente enemigo.

Condición: es la misma que en la regla de refuerzo. Hay que decir que esta regla además de atacar un continente para que el jugador enemigo no cobre ganancias extra, también sirve para evitar que el enemigo gane cuando el enemigo va a por ese continente porque sea presumiblemente su objetivo.

En este último caso hay que intentar que no consiga el continente, por lo que en la condición no se comprueba que tenga el continente entero, sino que se mira si la probabilidad que tiene de conquistarlo es alta (la primera condición está incluida en la segunda).

En esta regla y en el caso de que haya más de 2 jugadores, es decir, que los objetivos sean o bien 2 continentes o bien 24 territorios, también se comprueba si alguien está a punto de ganar viendo que tiene casi 24 territorios.

Acción: se ataca desde el territorio con la probabilidad más alta de ganar hasta un territorio de ese continente o en el segundo caso se ataca a los territorios de ese jugador que está a punto de conseguir 24 territorios.

Regla: Ir a por un continente fácil.

Condición: es la misma que en la regla de refuerzo

Acción: se ataca desde el territorio que más probabilidad tenga de ganar al territorio de ese continente.

Las siguientes 2 reglas sirven para cuando el jugador está en una mala situación de partida y con pocos territorios.

Regla: Atacar un territorio si la posibilidad de ganar es mayor que P.

Lo normal es atacar cuando la probabilidad de ganar es mayor o igual a 0.5, pero en el caso de que estemos perdiendo mucho hay que bajar esa probabilidad P. El valor de P depende de la situación de la partida, pudiendo ser 0.4 si la situación es mala o hasta 0.2 si la situación es muy mala. El objetivo de este ataque a la desesperada es la obtención de cartas para poder conseguir batallones extra.

Regla: Ir a por un continente cuya probabilidad de conseguirlo sea mayor que P.

Como en la regla anterior, lo normal para considerar un continente fácil de conquistar es que la probabilidad sea superior a 0.5. De hecho la regla anteriormente descrita de Ir a por un continente fácil se comprueba llamando al mismo método que esta regla, solo cambiando la probabilidad P a 0.5. Esta regla se aplica igual que la anterior, cuando se está en una mala situación y se intenta a toda costa conquistar un continente que nos dé refuerzos extra.

Finalmente y antes de enlazar con las reglas de movimientos, hay que hablar de otra decisión que hay que tomar. Cuando conquistamos un territorio, existe la posibilidad de pasar 0,1 ó 2 batallones al territorio conquistado siempre que se tengan esos batallones, además del que se pone por defecto ya que no puede haber territorios sin batallones. Para tomar esta decisión se comprueba lo siguiente:

-Se calcula la distancia mínima desde el origen hasta el enemigo más cercano, sea d1. Lo mismo para el territorio que acabamos de conquistar, sea d2. Si d1<d2, es decir, si el origen del ataque está más cerca del enemigo, no se pasan batallones. Por el contrario, si d1>d2, se pasan hasta 2 batallones al territorio conquistado. Si d1=d2, los batallones se quedan en el origen o se pasan al territorio conquistado según sea de fuerte ese enemigo más cercano.

Con esta regla se resuelven por ejemplo situaciones en las que se consigue un territorio aislado como Argentina en las cuales no tiene sentido pasar 3 batallones ya que Argentina está aislado y esos 3 batallones quedarían desperdiciados y obligaría a hacer un movimiento posterior innecesario.

4. Movimientos.

Las reglas de movimientos se basan en que es inútil tener una fuerza de batallones en un territorio que no esté pegado al enemigo, es decir, que esté aislado.

Regla: Continente en peligro

Condición: Lo primero que se comprueba es si tenemos algún continente conquistado, que tenga algún territorio aislado con más de un batallón y que el continente esté en peligro.

Acción: En este caso se mueven las tropas de ese territorio aislado hacia el enemigo mas cercano y entre los mas cercanos al mayor.

Regla: Mover desde territorio aislado

Condición: Si no tiene continentes conquistados o si los tiene pero no puede mover nada porque tenga 1 en cada territorio o si los tiene pero no corren peligro vamos a buscar el territorio con más batallones y que no tenga adyacentes enemigos (aislado) y hacemos movimientos hacia el enemigo más cercano y entre los mas cercanos al mayor.

Como se puede apreciar, la segunda regla incluye a la primera, pero las separamos porque consideramos más prioritario reforzar un continente amenazado.

Conclusiones Reglas:

En resumen, las reglas son independientes unas de otras. Así se facilita añadir, modificar o eliminar reglas sin que las otras se vean afectadas, es decir, se gana en flexibilidad / modularidad, pero se pierde eficiencia. Por ejemplo, en las reglas de ataque parejas a las de refuerzo, se tiene que comprobar otra vez la situación de la partida. Pero esta pérdida de eficiencia es necesaria, ya que hace falta comprobar la situación de la partida antes de aplicar cada regla, ya que como hemos comentado antes se puede empezar el turno atacando contra un continente enemigo y acabar atacando para ir a por tu objetivo.

También tiene una serie de desventajas, como la opacidad y dificultad de depuración. Es difícil examinar una partida y determinar qué acciones van a ocurrir (depende del orden de las reglas). No hay un flujo de control claro.

La división de la inteligencia en reglas hace que cada regla individual sea fácilmente tratable, pero se pierde la visión global.

Clases principales:

InteligenciaSistemaExperto: subclase de Inteligencia. Implementa los métodos abstractos de Inteligencia refuerzo, ataque y movimientos. La inteligencia del sistema experto depende del objetivo, así que tenemos las subclases Inteligencia2Continentes e InteligenciaNTerritorios. Esta clase implementa el método aplicarReglaAtaque, que es el motor por el cual se aplican las reglas en un determinado orden. El motor para aplicar las reglas de refuerzo es el mismo para las 2 subclases, pero lo hemos implementado en ellas en vez de en esta. Esto es debido a que para obtener estadísticas de partidas hemos hecho diferentes órdenes de refuerzo (así como de ataque), pero sólo nos interesaba hacerlo para 2 jugadores, es decir, para InteligenciaNTerritorios.

Inteligencia2Continentes: subclase de InteligenciaSistemaExperto. Se crea cuando el objetivo es el de conquistar 2 continentes. Implementa el motor para aplicar las reglas de refuerzo con el mejor orden de prioridad que hemos conseguido en función de las estadísticas.

InteligenciaNTerritorios: subclase de InteligenciaSistemaExperto. Esta clase sirve tanto para el caso de que el objetivo sea 24 territorios como para cuando el objetivo sea conquistar el mundo (en total son 42 territorios), la única diferencia es el número N de territorios que hay que conquistar, 24 ó 42. Aquí se implementa el motor para aplicar las

reglas con diferentes órdenes en función de un parámetro pasado en la constructora de JugadorMáquina. Esto nos ha servido para generar estadísticas de qué orden es el mejor.

Detalles de implementación

De todas las reglas descritas anteriormente, sólo las reglas *A punto de ganar* y *A por mi objetivo*, tanto las de refuerzo como las de ataque son específicas de cada objetivo. Así que esas reglas se implementan en las subclases Inteligencia2Continentes e InteligenciaNTerritorios. Las demás reglas son comunes a todos los objetivos, por lo que se implementan en la clase InteligenciaSistemaExperto.

Una vez implementadas las reglas tenemos que comprobar qué orden es el mejor a la hora de aplicarlas, es decir, la prioridad que tiene cada regla. Para ello implementamos un método que según un valor llame a las reglas en un orden o en otro. Así, cambiando simplemente este valor, podremos crear una estrategia completamente distinta.

Este orden intenta tener en cuenta de igual forma las reglas de refuerzo y las de ataque, es decir, si la primera regla de refuerzo es ir a por un continente enemigo, entonces la primera regla de ataque será la de atacar continente enemigo. Pero hay reglas de refuerzo que no se corresponden con ninguna de ataque, como la de reforzar un continente amenazado, que no tiene porqué suponer un ataque posterior.

Resumiendo las reglas y numerándolas para referirnos a ellas más fácilmente, quedan así (esta numeración sigue el orden que fijamos desde el principio, orden que pensamos que sería el mejor y que finalmente tras las pruebas hemos corroborado):

Refuerzo:

1. Reforzar estando a punto de ganar.
2. Reforzar continentes amenazados
3. Reforzar contra continente enemigo
4. Reforzar continente fácil
5. Reforzar para ir a por mi objetivo

En cuanto a las de ataque, las reglas 1 y 2 se aplican al principio, aunque si se aplicaran al final no supondría ningún cambio, ya que sólo se aplican en caso de ir perdiendo y en ese caso no se aplicaría ninguna de las otras reglas. Conclusión: da igual ponerlas al principio que al final.

Ataque:

1. Atacar continente con probabilidad baja.
2. Atacar territorio con probabilidad baja.
3. Atacar estando a punto de ganar.
4. Atacar continente enemigo
5. Atacar para conseguir continente fácil
6. Atacar para ir a por mi objetivo

INTELIGENCIA RED NEURONAL.

Introducción

En un principio, nuestra aplicación iba a constar solo de un Sistema Experto. Pero luego nos dimos cuenta de que nuestro proyecto sería más interesante si incluyésemos otros tipos de inteligencia artificial. Así surgieron ideas de usar Redes Neuronales y Algoritmos Evolutivos.

Nos pusimos manos a la obra y hay que decir que por falta de tiempo decidimos que sólo podríamos usar una de las 2 tecnologías, y nos decantamos por las Redes Neuronales.

La visión inicial

Hay que decir que ninguno de los 3 componentes del grupo teníamos muchas nociones de Redes Neuronales en un principio.

Inicialmente nuestra idea era conseguir un sistema que pudiese jugar usando Redes de la misma forma que usamos el Sistema Experto. Esto quiere decir que pudiésemos jugar partidas de hasta 6 jugadores siendo 5 de ellos manejados usando Redes Neuronales.

Luego nos dimos cuenta de que generar ejemplos para situaciones de la partida en la que hubiese tantos jugadores podía resultarnos bastante complejo, y que incluso consiguiendo generar ese tipo de ejemplos el número de situaciones o patrones que la Red debería identificar se multiplicaría haciendo así más complicada la identificación de dichos patrones por parte de la Red.

Así que decidimos que reduciríamos el problema a una partida de 2 jugadores. Además de esta forma también simplificábamos todo el tema de los objetivos, ya que en una partida de tan sólo 2 jugadores el objetivo de cada uno se reducía a conquistar el mundo.

Reutilización de código

Ciertas funcionalidades de nuestro proyecto no han tenido que ser implementadas desde cero. Esto nos ha proporcionado un ahorro de esfuerzo y tiempo considerable.

Una de estas funcionalidades fue la red neuronal. La implementación de la red nos la proporcionó nuestro coordinador gracias a lo cual nos despreocupamos totalmente de su programación. Simplemente, tuvimos que estudiar el tipo de ejemplos y la manera de proporcionárselos así como la interpretación de las salidas de la red.

La red neuronal: Concepto y Diseño

Una red neuronal no es otra cosa que una función que aprende por "fuerza bruta". Esto significa que la red aprende a base de entrenar sobre unos mismos ejemplos un número determinado de veces, y que cuanto mayor sea ese número de veces más exacta será esa función. La función se calcula como la suma de los productos de cada entrada a un nodo el peso de dicha entrada y de aquí se obtendrá un **valor**. Además en nuestro caso concreto los nodos son sigmoides por lo que ese **valor** deberá ser "filtrado" de la siguiente forma:

$$\textbf{res} = 1. / (1. + \exp(\textbf{-valor}))$$

res será la salida del nodo que a su vez sirve de entrada al siguiente.

La aplicación que nos proporcionó nuestro coordinador consistía en un reconocedor de imágenes. Las imágenes tenían una resolución de 30x32 píxeles que se usaban como entradas. Y las salidas hacían referencia al nombre de la persona que aparecía en la imagen, si llevaba gafas o no, si estaba triste o contenta, etc.

Nuestra aplicación, en cambio, representa las entradas como la situación del tablero en un momento dado haciendo referencia al estado de los 42 territorios, quién los gobierna y cual es su ocupación. En cuanto a las salidas también son 42 y hacen referencia a los territorios. Concretamente nos indican cual es el territorio que la red ha decidido como el que más se ajusta al patrón de entrada.

De esto se deduce que tuvimos que eliminar ciertas clases de la aplicación inicial y quedarnos solamente con las que nos servían. Nos deshicimos de todas las clases relacionadas con el tratamiento de imágenes y nos quedamos con las siguientes:

- **Red**: Esta clase implementa la red neuronal. Sus atributos son el número de entradas, de salidas y de ocultas. La tasa y el momento y por último 2 vectores de Sigmoides. Sus métodos más importantes son: *salida* que nos devuelve la salida que devolvería la red a un vector de entrada dado, *propagaError* que como su nombre indica transmite el error de la salida a las capas intermedias y *actualizaPesos* que modifica los pesos de la red tras la última iteración de entrenamiento.

- **Sigmoide**: Esta clase implementa los nodos intermedios y los nodos salidas de la red. Lo más importante de esta clase es que almacena un vector de doubles que son los pesos de ese nodo.

- **Entrenador**: Esta clase es la encargada de proporcionar a la red un entrenamiento. Sus atributos son la red a entrenar y la ruta de la que debe obtener los ejemplos para su entrenamiento. Esta clase dispone de un método llamado *iteración* que lleva a cabo el entrenamiento de una red aplicando una "pasada" a todos los ejemplos de esa red, actualizando los pesos y propagando el error. Además este método nos devuelve el valor del error absoluto y de la precisión de la red en esa iteración. Es importante comentar que a diferencia de la aplicación inicial en la que los ejemplos de entrada a la red eran imágenes, en esta

clase se realiza la obtención de los ejemplos a través de un método (*leeEjemplos*) que los va leyendo de un fichero de texto.

- **Ejemplo**: Esta clase se usa para codificar la estructura de un ejemplo que no son más que 2 vectores de doubles llamados *entradas* y *salidas* respectivamente.

Además de estas clases propias de la aplicación inicial, también cabe destacar la existencia de la clase *InteligenciaRedNeuronal*.

- **InteligenciaRedNeuronal**: esta clase hereda de la clase *Inteligencia* y se encarga de implementar el comportamiento de los jugadores cuya inteligencia esta basada en redes neuronales. Esta clase dispone de 5 atributos de la clase *Red* y otros 5 de la clase *Entrenador*. Como se explicará a continuación se necesitan distintas redes para codificar el comportamiento de la red. Unas se usan para la fase de Refuerzo, otras para la de Ataque y otras para la de Movimientos. La clase *InteligenciaRedNeuronal* dispone de un método llamada *ejecuta* al que le pasamos el número de iteraciones y entrena a la red correspondiente ese número de veces llamando al método *iteración* de la misma. Además esta clase dispone de métodos para guardar la red y cargarla, con lo que nos ahorramos tiempo de entrenamiento.

El Problema: ¿Cuántas Redes Usar?

El número de Redes que debíamos usar para solucionar un problema tan complejo como un turno en una partida de Risk también nos dio bastantes quebraderos de cabeza.

En un principio se pensó en usar 3 Redes, una para cada fase de un turno:

- Una para la fase de Refuerzo.
- Una para la fase de Ataque.
- Una para la fase de Maniobra.
-

Sin embargo, era imposible cubrir las necesidades de cada fase del juego tan solo con una Red, ya que se debían tomar varias decisiones en cada una. Y por cada una de esas decisiones necesitaríamos una Red independiente.

Así por ejemplo para la fase de **Refuerzo** hubiésemos necesitado:

- Una para la decisión de canjear cartas por ejércitos o no.
- Una para decidir cual de nuestros territorios reforzar.
- Una para decidir con cuantos batallones reforzar el territorio.

En la fase de **Ataque**:

- Una para decidir desde qué territorio atacar.
- Una para decidir a qué territorio atacar.
- Una para decidir, en caso de ganar, cuantos ejércitos pasar al territorio conquistado.

En la fase de **Movimientos**:

- Una para decidir desde que territorio mover ejércitos.
- Una para decidir a qué territorio mover los ejércitos.
- Una para decidir cuantos ejércitos mover.

En total 9 Redes. De estas 9 tan solo hemos decidido implementar aquellas que daban como resultado un territorio. Así han quedado:

Fase de **Refuerzo**:

- Una para decidir cual de nuestros territorios reforzar.

En la fase de **Ataque**:

- Una para decidir desde qué territorio atacar.
- Una para decidir a qué territorio atacar.

En la fase de **Movimientos**:

- Una para decidir desde qué territorio mover ejércitos.
- Una para decidir a qué territorio mover los ejércitos.

El resto de decisiones que debemos tomar en cada fase lo haremos usando los métodos propios del Sistema Experto. Aunque se han pensado algunas soluciones para ellas:

- Para la red de canjeo se pensó en elaborar ejemplos en los que las entradas reflejaran la situación de la partida en un determinado momento y las salidas fuesen 2: **sí** o **no** se debe canjear cartas.

- Para las redes que deciden un número de ejércitos para reforzar lo que se pensó era generar un ejemplo distinto para cada ejército que se reforzara en lugar de uno solo, es decir, realizar la fase de Refuerzo paso por paso o “batallón por batallón”. De igual manera se pensó hacer para decidir el número de batallones a mover en las fases de Ataque y Maniobra.

Otro problema: ¿Cuándo dejar de atacar?

En el Sistema Experto esto sucede cuando la probabilidad de realizar un ataque y salir vencedor esta por debajo de un valor. Sin embargo, en la red es algo más complejo,

ya que el comportamiento de la red está basado en la identificación de patrones y dar valores a las salidas según ese patrón.

Lo que inicialmente pensamos fue elaborar ejemplos en los que todas las salidas de la red fuesen 0.1. Es decir, si a lo largo de una fase de ataque lograbas identificar un patrón cuyas salidas fuesen todas 0.1 se dejaba de atacar. Sin embargo esto no va a ocurrir nunca porque el cálculo de las salidas de la red nunca dará una salida completa a 0.1, dará una serie de valores reales entre 0.1 y 0.9 y de esos escogeremos el mayor.

De modo que para solucionar esto simplemente decidimos realizar 3 ataques por turno, siempre que los ataques que calcula la red neuronal sean coherentes.

Fases en el uso de una Red Neuronal

Para usar una Red Neuronal correctamente esta debe pasar por varias fases previas a su utilización. Estas fases son las siguientes:

- Generación de ejemplos.
- Entrenamiento de la red.
- Uso de la red.

Ahora vamos a ir explicando correctamente cada una de ellas.

Generación de Ejemplos:

En esta fase teníamos que elaborar ejemplos que luego la red debería identificar como patrones.

En un principio se pensó elaborar estos ejemplos de forma manual como un fichero de texto en el que aparecieran 42 entradas que representaban la situación de cada territorio en la partida en un momento dado y que encerrasen información sobre a quien pertenecía dicho territorio y sobre su ocupación en ejércitos. También aparecerían 42 salidas todas con un valor de 0.1 excepto una con 0.9 que sería la correcta.

Como ya hemos dicho esto se empezó haciendo de forma manual intentando englobar las situaciones más representativas que se podían dar a lo largo de una partida como por ejemplo: *Nos queda un solo territorio para ganar, nos falta un solo territorio para conquistar un continente, etc.*

Sin embargo, esta metodología conllevaba un gran esfuerzo. De modo que se pensó que ya que lo que deseábamos era una red que actuase siguiendo las mismas reglas que el Sistema Experto, lo mejor sería utilizar el propio sistema Experto para generar los ejemplos. Y para ello lo que se hizo fue poner a jugar al Sistema Experto contra un jugador con inteligencia Aleatoria durante un número de partidas e ir archivando las decisiones que tomaba el Sistema en ficheros de texto.

La codificación de los ejemplos sería la siguiente: 42 entradas que como ya hemos dicho representaban la situación de los territorios del mapa. Las entradas serían un valor real entre 0.1 y 0.4 para los territorios del jugador Aleatorio y entre 0.6 y 0.9

para los territorios de la red neuronal. De esta forma se codificaba la posesión del territorio. Para codificar la ocupación del territorio lo que se hizo fue establecer un número máximo de batallones por territorio. Se cogería el rango de 0.1 a 0.4 y de 0.6 a 0.9 (es decir 0.3) y se dividiría entre ese número máximo obteniéndose un **valor**. Y por cada batallón que ocupase el territorio se sumaría o restaría ese **valor** a 0.6 y a 0.4 respectivamente.

De esta manera si el jugador aleatorio tenía el número máximo de batallones (o más de ese número) en un territorio su entrada valdría 0.1 y si solo tenía 1 valdría 0.4. De igual forma para la red neuronal valdría 0.6 si solo tenía un batallón y 0.9 si tenía el número máximo (o más).

Las salidas permanecerían igual, es decir 0.9 para el territorio salida y 0.1 para el resto.

Este podría ser un ejemplo:

Entradas:	Salidas:
0.6	0.1
0.336842	0.1
0.384211	0.1
0.6	0.1
0.384211	0.1
0.6	0.1
0.352632	0.1
0.6	0.1
0.384211	0.1
0.663158	0.1
0.6	0.1
0.384211	0.1
0.6	0.1
0.368421	0.1
0.694737	0.1
0.384211	0.1
0.6	0.1
0.368421	0.1
0.4	0.1
0.6	0.1
0.4	0.1
0.6	0.1
0.368421	0.1
0.678947	0.1
0.352632	0.1
0.6	0.1
0.6	0.1
0.368421	0.1
0.6	0.1
0.4	0.1
0.6	0.1
0.368421	0.1
0.4	0.1
0.726316	0.9
0.4	0.1
0.6	0.1
0.6	0.1
0.384211	0.1
0.4	0.1
0.663158	0.1
0.6	0.1
0.4	0.1

Entrenamiento de la Red Neuronal:

Para llevar a cabo el entrenamiento de una red necesitamos que esta disponga de un entrenador. Por ello, en la clase *InteligenciaRedNeuronal* además de tener las 5 redes, tendremos un entrenador para cada red.

En la clase *InteligenciaRedNeuronal* tenemos un método llamado *entrenarRedes* en el que creamos las 5 redes inicializándolas con el número de entradas, de salidas y de capas ocultas. Después se crean los 5 entrenadores pasándoles como parámetro la red a la que deben entrenar y la ruta de donde deben leer los ejemplos para llevar a cabo el entrenamiento.

Por último, para generar el entrenamiento propiamente dicho, tenemos un método llamado ***ejecuta*** al que se la pasa el entrenador que deseamos entrenar y un entero *iteraciones* que me indica cuantas veces queremos que el entrenador ejecute su método *iteración* que lo que hace es entrenar a la red una vez sobre unos ejemplos dados y devuelve la precisión y el error con la que la red reconoce dichos ejemplos. Lo ideal sería que el método *iteración* se invocase tantas veces como fuese necesario hasta que el error y la precisión devueltos estuviesen por debajo y por encima respectivamente de un valor dado.

Cuando se estime que la precisión y el error de la red son los adecuados habrá finalizado el entrenamiento de la red.

Uso de la red:

Una vez finalizadas las fases previas de generación de ejemplos y de entrenamiento de la red ya estamos en disposición de utilizar la red para tomar decisiones.

Hay que decir que las fases previas solo se realizarán si están a false los atributos de la clase *juego ejemplosgenerados* y *redesentrenadas*. El motivo de esto es que estas fases pueden tardar un periodo de tiempo prolongado en acabar, y sería poco eficiente realizarlas cada vez que se inicie el juego.

Por otra parte la red neuronal es una estructura cuyos valores pueden ser almacenados y cargados cuando se desee ahorrándonos así las fases de generación de ejemplos y de entrenamiento.

Para usar la red neuronal simplemente debemos transformar la situación actual en la partida en una estructura que pueda ser usada como entrada a la red. Luego debemos pasar esta estructura como parámetro de entrada al método *salida* de la red neuronal el cual nos devolverá otra estructura de salida y obtener el valor máximo de esa estructura. Esa será la salida deseada.

Conclusiones obtenidas sobre la red neuronal

En este apartado se realizará un estudio de los resultados obtenidos en el uso de las redes neuronales y si estos han sido satisfactorios o no realizando pruebas sobre las mismas.

En un principio nuestra idea era entrenar una misma red varias veces modificando cada vez tanto el número de iteraciones como el de ejemplos. Así por ejemplo queríamos empezar con un entrenamiento "suave" de unos 100 ejemplos con 1000 iteraciones.

Más tarde iríamos incrementando el número de ejemplos y el de iteraciones hasta conseguir redes que se aproximaran con un cierto grado de exactitud al comportamiento del Sistema Experto.

La realidad es que la complejidad del problema del entrenamiento de las redes se multiplicaba de una manera desorbitada a medida que incrementábamos el valor de los parámetros. Nuestro objetivo era alcanzar una red que estuviese entrenada usando un número aproximado de 5000 ejemplos y 5000 iteraciones. Sin embargo debido a la gran complejidad del problema y a la falta de tiempo tuvimos que conformarnos con una implementación más realista y, por otra parte, más limitada.

Finalmente decidimos entrenar las redes con un número de 100 ejemplos y 2500 iteraciones, y aún así el entrenamiento tuvo una duración bastante prolongada (del orden de horas). Y ni siquiera logramos resultados satisfactorios al final de dicho entrenamiento ya que ninguna de las redes había alcanzado una precisión del 90%.

Una vez que finalizamos los entrenamientos de las redes nos dispusimos a probarlas en la aplicación. Los resultados fueron bastante decepcionantes debido a un entrenamiento llevado a cabo sobre un número tan bajo de ejemplos diferentes (tan solo 100). Como era de esperar las situaciones que la red era capaz de distinguir y responder con éxito estaban muy limitadas sobre todo teniendo en cuenta que el espacio de ejemplos que se puede extraer de un problema tan complejo como el RISK es muy disperso.

Como conclusiones finales debemos comentar que el uso de redes neuronales para un problema de esta complejidad conlleva un esfuerzo considerable en la fase de entrenamiento de las redes. Para obtener resultados satisfactorios el número de ejemplos de entrada a la red debe ser muy elevado lo que implica a su vez un incremento en el número de iteraciones necesarias para reconocer dichos ejemplos.

Todo esto dará lugar a un periodo de entrenamiento de redes muy prolongado y complejo, aunque una vez finalizado el mismo los resultados obtenidos podrían acercarse con bastante exactitud al comportamiento ideal.

11.3. Gráficos

Descripción:

La principal interfaz de nuestro proyecto consta de una clase de tipo TForm (Formulario del C++Builder) en la que se va a mostrar todo lo necesario en el transcurso de la partida. Este formulario (clase llamada TInterfaz) está elaborado mediante técnicas con OpenGL, usando los conocimientos aprendidos en la asignatura Informática Gráfica. Antes de llegar a esa interfaz donde veremos la partida, hay un menú de juego en el que podemos elegir las opciones y parámetros de la partida.

Además, en el desarrollo de la partida, podrán aparecer nuevos formularios para gestionar las cartas, el ataque o los movimientos. Estos formularios son normales, hechos con botones, scrollBars, paneles, etc.

Primero veremos la estructura de la parte gráfica:

Clases principales:

TFormMenu: esta es la primera ventana que nos aparece al ejecutar el proyecto. Consiste en un menú de juego en el que podemos desplazarnos con teclado. Tiene una serie de opciones, accesibles pulsando la tecla Enter. Está hecho con OpenGL, ya que inicialmente queríamos hacer algo más vistoso, pero hemos tenido que dejarlo por falta de tiempo.

TformMenuOpciones: parecido a la clase anterior, sólo que esta tiene opciones que se pueden cambiar mediante teclado. Aquí se pueden cambiar los parámetros del juego.

TInterfaz: esta clase es el formulario donde se verá la partida. Se crea cuando se selecciona nueva partida en el menú principal. Además se crea el objeto de la clase Juego que será el motor de la partida. Además del panel con el mapa, habrá una especie de barra de herramientas: habrá una foto de una brújula para explorar la escena, fotos de lupas para hacer zoom in y zoom out. Para las partidas en las que juegue un humano, se hará visible un panel que contiene una serie de botones útiles para el desarrollo de su turno. Así para la fase de refuerzo se podrá ver en un panel el número de ejércitos por asignar. Para asignarlos se deberá pinchar sobre el territorio, pero previamente para poner de 1 en 1 o de 5 en 5 o de 10 en 10 hay que pinchar sobre las fotos de la derecha de un soldado, un caballo y un cañón.

También habrá un botón para pasar el turno cuando se desee, siempre que no queden ejércitos por asignar. También incluimos un botón para poder ver las cartas que tienes en cualquier momento, otro botón para ver el objetivo actual y otro botón que muestra un panel con información sobre los jugadores y sus respectivos colores y el color del jugador que posee el turno.

Escena: se crea en la constructora de la clase Juego. Contiene información sobre el ancho y alto del puerto de vista o tamaño de la pantalla donde se visiona, necesario para explorar el mapa y hacer zoom. Además necesita la información de la clase Mapa (la lista de territorios) para poder crear la clase InterfazMapa.

InterfazMapa: usando la lista de territorios crea por cada territorio un objeto de la clase InterfazTerritorio, que contendrá la información necesaria para dibujar los territorios en el formulario.

InterfazTerritorio: consiste en una lista de listas de líneas (ver detalles de implementación).

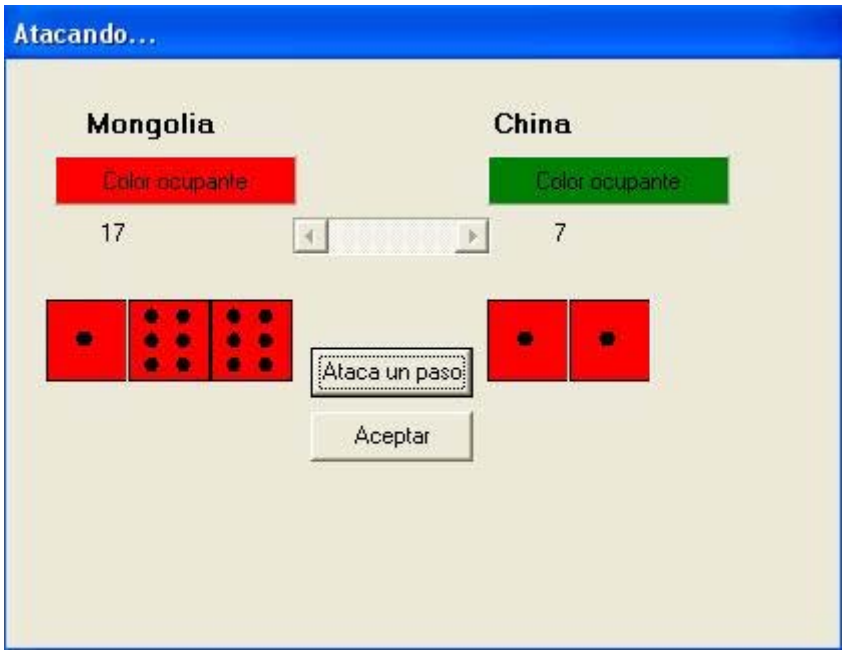
Además tiene un puntero al objeto de la clase Territorio al que representa para obtener información sobre el color del ocupante, el nombre del territorio y el número de batallones que tiene, todo ello necesario a la hora de pintarlo en la pantalla.

TVentanaAtaque: es una ventana que se abre cuando el jugador humano ataca de un territorio a otro. Se muestra información de los territorios origen y destino, como sus colores, nombres y número de batallones. También hay un botón "Atacar un paso" que simula la tirada de dados correspondiente al número de batallones que hay tanto en origen como en destino. En la ventana de ataque 1 se muestra un ataque antes de hacerse, nada más abrirse la ventana. En la ventana de ataque 2 se muestra el resultado de pulsar una vez el botón Ataca un paso. Como se puede ver, la tirada de dados ha favorecido al atacante, por lo que el defensor pierde 2 batallones y se queda con 7.

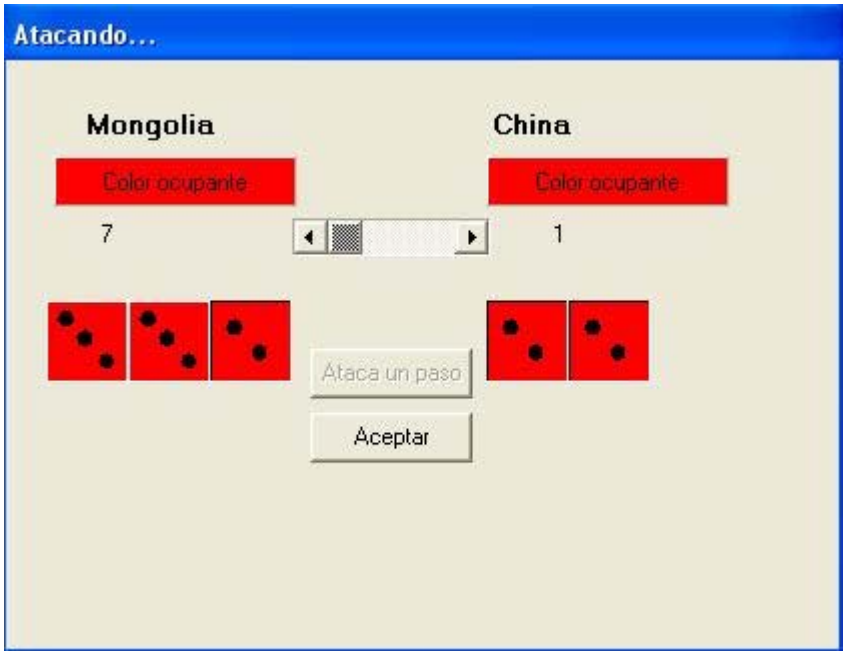
Cuando se ha conquistado el territorio defensor se activa el scrollBar para poder mover de 0 a 2 batallones desde el origen al destino, como se puede ver en la Ventana de ataque 3. Después de esto, se pulsa el botón Aceptar para cerrar la ventana.

Ventana de ataque 1

Ventana de ataque 2

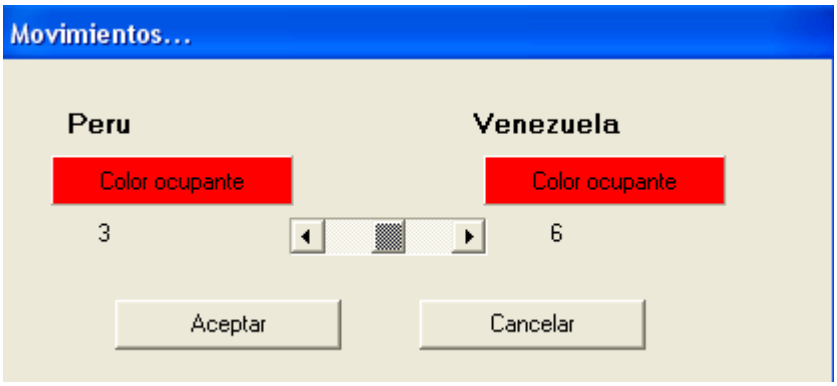


Ventana de ataque 3



TVentanaMovimientos: es la ventana que se abre cuando el jugador humano hace movimientos. Se muestra el territorio origen y el destino del movimiento, así como el número de batallones que tiene cada uno y se habilita un scrollBar para poder pasar batallones de un sitio a otro, pudiéndose rectificar el número de batallones que se mueven hasta que no se da al botón Aceptar. También se puede Cancelar si se ha equivocado y no quiere hacer efectivo el movimiento

Ventana de movimientos

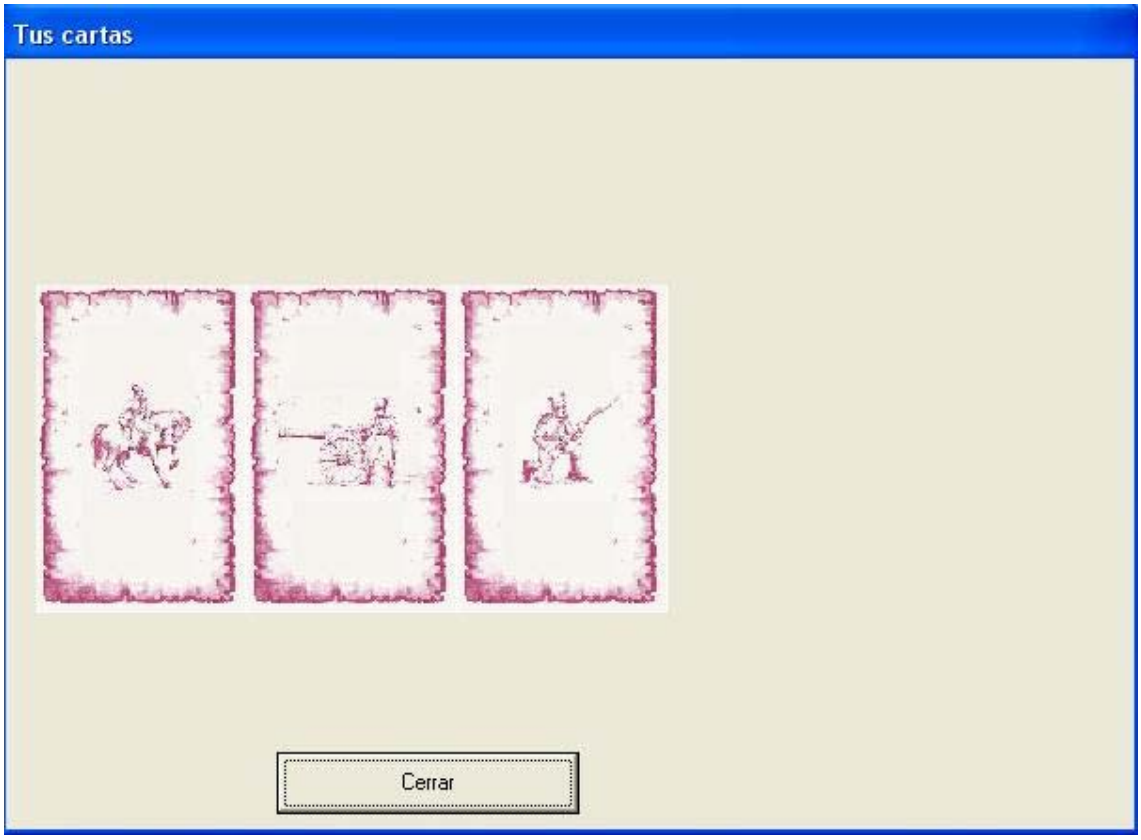


TFormCartas: es la ventana que se abre cuando el usuario pulsa el botón de Ver Cartas situado en la parte inferior del formulario principal, en la que se visualizan en forma de fotos las cartas que tienes en posesión. Esta ventana se puede abrir pulsando ese botón a lo largo del turno, o se puede abrir sola al principio del turno cuando se detecta que se tienen cartas suficientes para realizar un canjeo. En el primer caso el botón Canjear está oculto y sólo se puede Cerrar (ver Ventana de cartas 1). En el segundo caso, se puede Cerrar sin canjear o pulsar el botón Canjear. En este segundo caso se incluye información sobre qué cartas se pueden canjear y cuántos batallones te dan en cada caso.

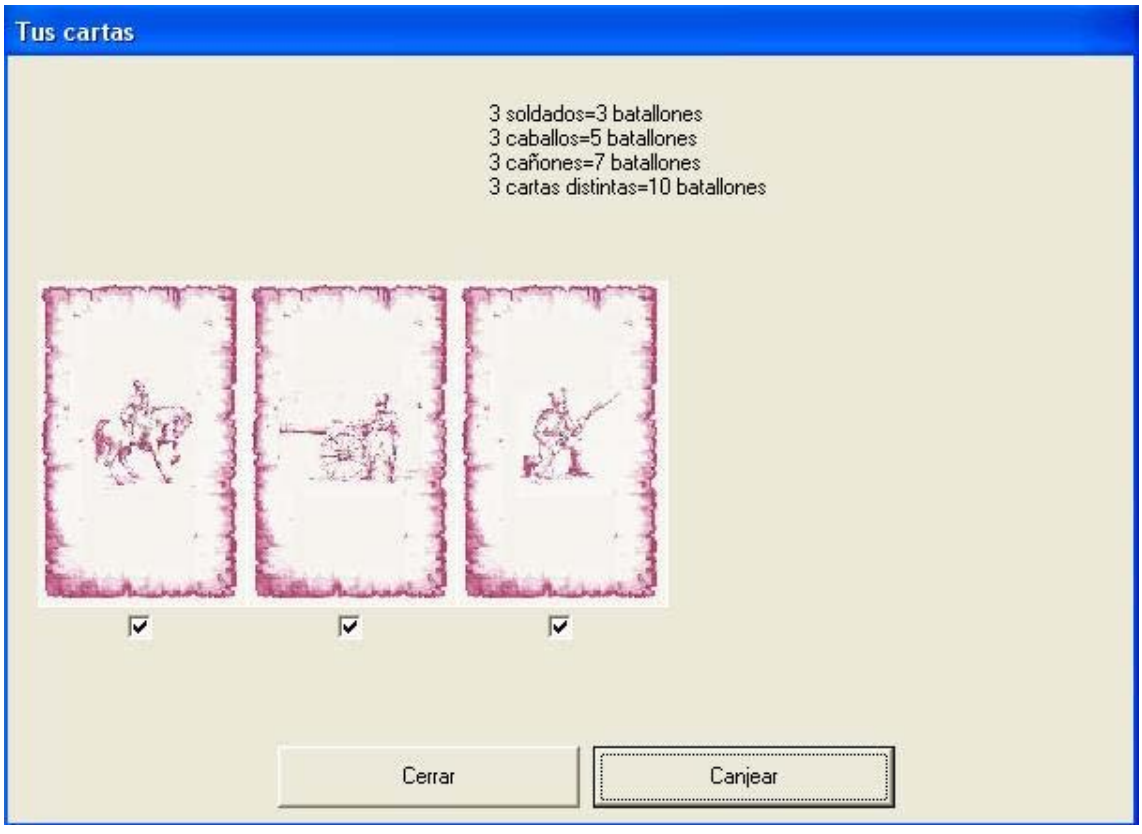
En el caso de querer canjear, hay que seleccionar las imágenes de las cartas activando el *CheckBox* que hay debajo de cada una (ver Ventana de cartas 2).

Si no se seleccionan las cartas adecuadas y se pulsa Canjear, el sistema avisa de que no se puede canjear. También avisa si se intenta cerrar la ventana cuando se tienen 5 cartas, en cuyo caso hay que canjear por obligación.

Ventana de Cartas 1



Ventana de Cartas 2



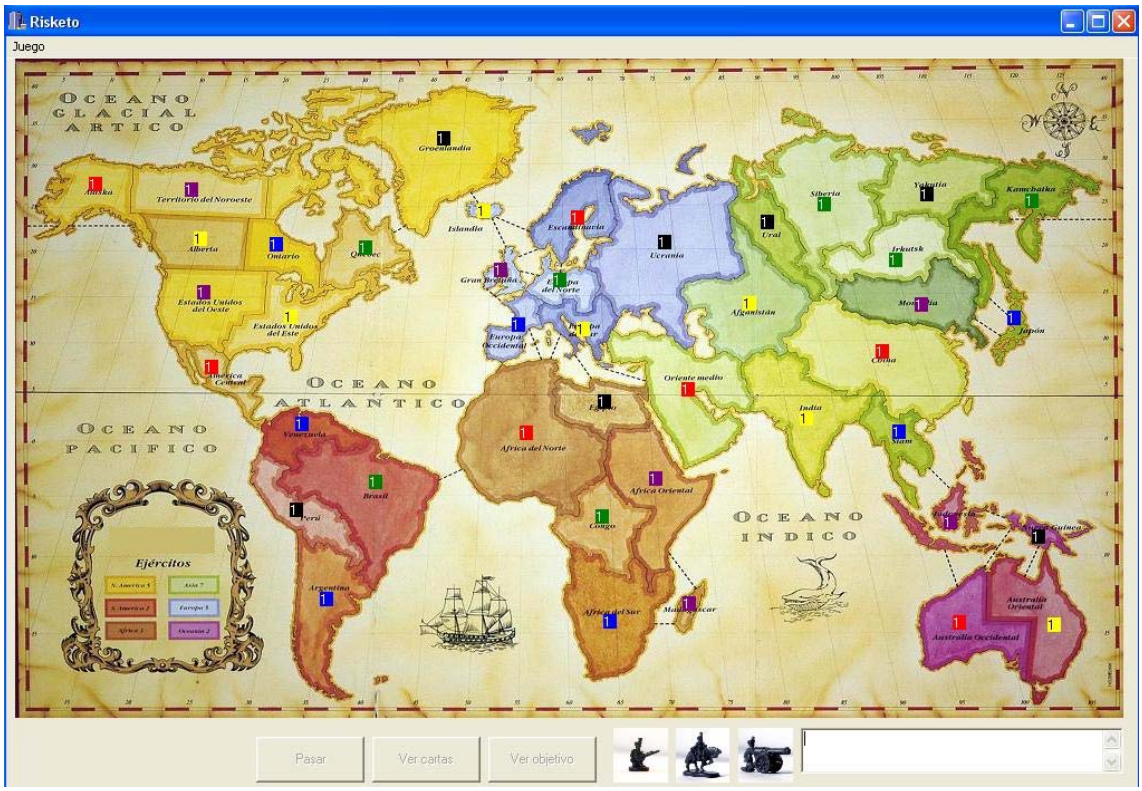
Detalles de la implementación y evolución de la interfaz:

La principal cuestión a resolver en cuanto a la interfaz gráfica es como se seleccionarían los territorios para realizar un refuerzo, ataque o maniobra.

Inicialmente, en la primera interfaz que hicimos (ver fotos), se hacía guardando por cada territorio una posición con sus coordenadas x e y. Esa posición se introdujo con la práctica de LP3 antes mencionada.

Al pulsar con el ratón en la imagen del tablero se comprobaba que las coordenadas de la pulsación del ratón estuvieran cerca de las coordenadas de algún territorio, es decir, que estuvieran dentro del rango del territorio. El problema es que ese rango no podía ser muy grande, ya que hay territorios pequeños y juntos que solaparían sus rangos si éstos fueran grandes. Por lo tanto el rango era de unos 5 píxeles, lo que en territorios grandes como Groenlandia era un pequeño espacio y había que estar buscando el punto donde poder pulsar. Para saber donde se podía pulsar lo que hacíamos era cambiar el cursor del ratón de un puntero a una mano cuando se estaba dentro del rango de un territorio.

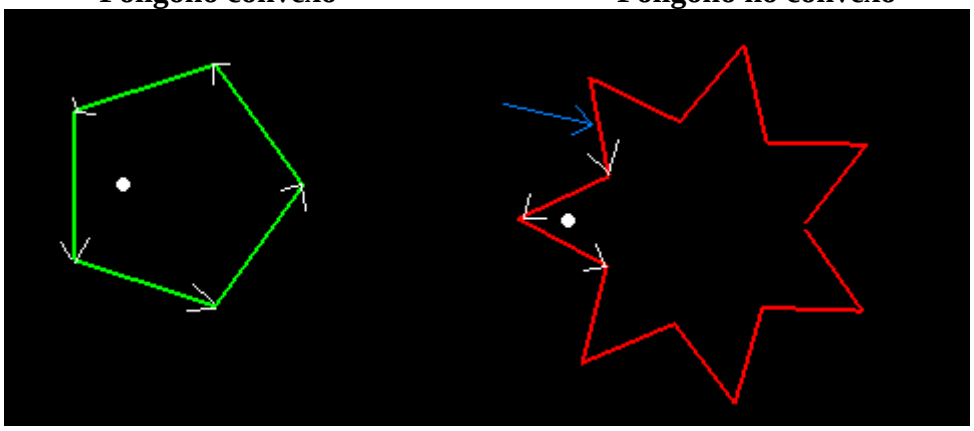
Interfaz inicial



Para realizar la nueva interfaz, tuvimos en cuenta un concepto aprendido en Informática Gráfica por el que se detecta si un punto está dentro de un polígono convexo, usando el producto vectorial, con el cual podemos comprobar si el punto está a la izquierda de todas las líneas que forman el polígono. El problema surge cuando el polígono no es convexo. Esto significa que un punto dentro del polígono no tiene porqué cumplir la propiedad de quedar a la izquierda de todas las líneas del polígono. Como se ve en la figura de abajo, un punto interior a un polígono convexo siempre cumple que queda a la izquierda de todas las aristas, mientras que en un polígono no

convexo hay puntos internos que no quedan a la izquierda de todas las aristas. Véase la arista señalada por la flecha azul, el punto interior queda a su derecha.

Polígono convexo **Polígono no convexo**

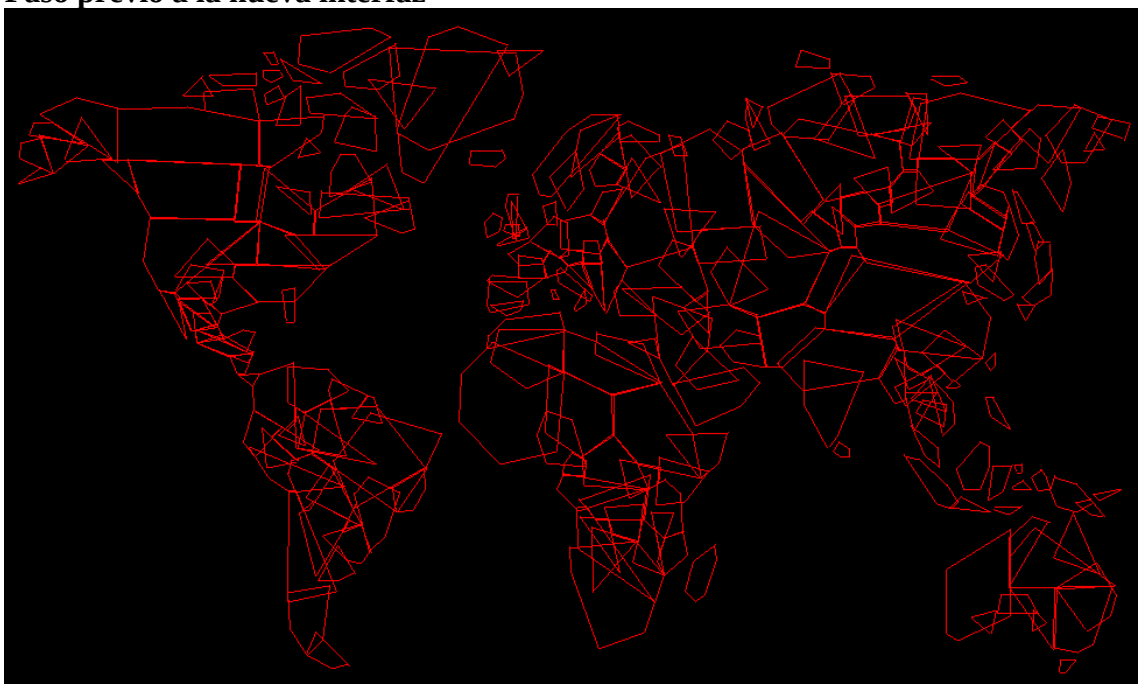


Volviendo a nuestro tema, se puede comprobar que todos o casi todos los territorios son polígonos no convexos. La solución era hacerlos cuadrados pero quedaría muy pobre la interfaz. La solución tomada fue la de por cada territorio guardar la lista de polígonos convexos que lo forman (de ahí la estructura de lista de listas de líneas, cada polígono es una lista de líneas).

Estos polígonos convexos se introdujeron “a mano” en un editor de escenas tipo Paint que hicimos en Informática Gráfica, práctica 2, cargando la foto del tablero de fondo para tener una guía de por donde ir dibujando las líneas.

Después de un trabajo laborioso el mapa tenía el siguiente aspecto.

Paso previo a la nueva interfaz



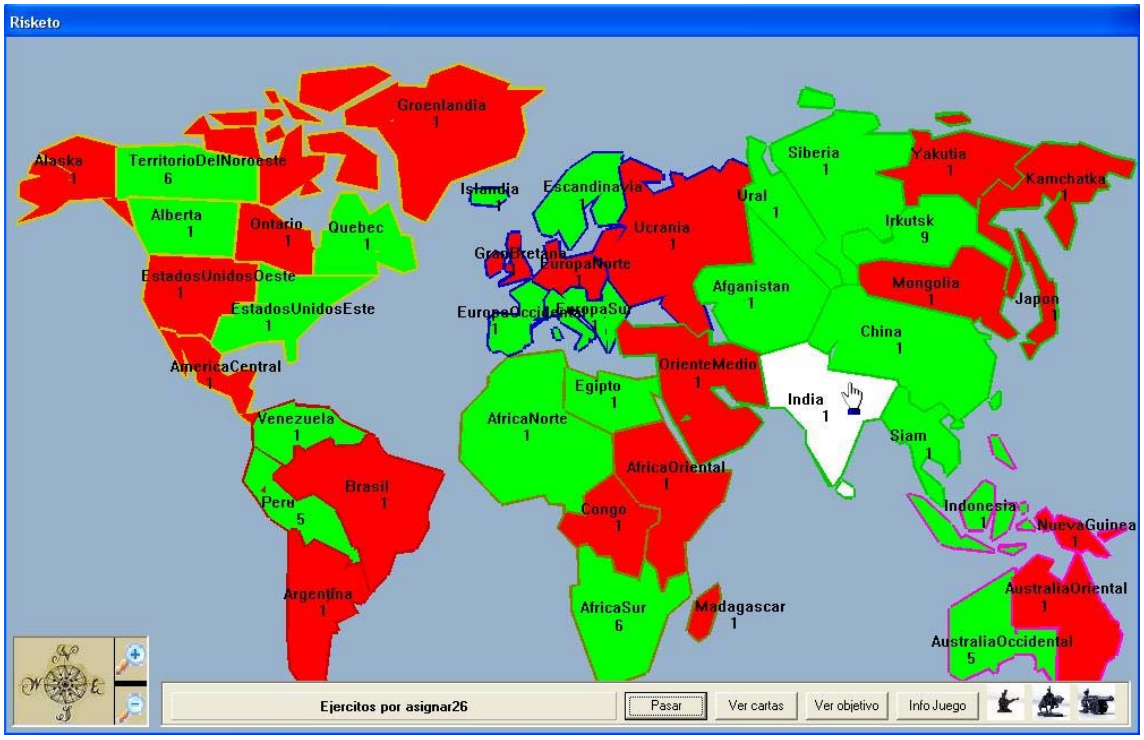
Toda esta información se guardó en un fichero, y se lee al crear la clase InterfazMapa.

Ahora para detectar la pulsación se recorre la lista de polígonos convexos que forman el territorio y utilizamos el método estáDentro reutilizado de la práctica 2 de

IGR. Para avisar del territorio que va a ser detectado al pulsar el ratón, además de cambiar el cursor del ratón a una mano, también pintamos el interior del territorio de color blanco. En la foto de abajo se ve que la India está seleccionada.

Para pasar del paso previo hasta la nueva interfaz, solamente hay que pintar el interior de los polígonos y así se ocultan todas las líneas menos las de los bordes, las cuales pintamos con diferentes colores en función del continente y con más grosor para que se vean bien.

Nueva interfaz



Se conservaron los botones de Pasar turno, Ver cartas y Ver objetivo, y las fotos del soldado, caballo y cañón para la fase de refuerzo. Se introdujo otro botón InfoJuego en el cual al pasar el ratón por encima te saca un panel de información sobre los colores de cada jugador, útil para saber que tú eres el color rojo cuando juega un humano. También te muestra el color del jugador al cual le toca jugar. Aquí hay que apuntar que cuando juegan las máquinas el juego está dentro de un bucle, así que no se detecta ni el movimiento ni la pulsación del ratón, así que el color del panel informando del turno siempre que lo puedas ver estará del color del humano, color rojo.

Se incluyeron todos estos botones anteriores en un panel que sólo está visible cuando juega un humano. Cuando juegan sólo máquinas no tenía sentido que se pudiera ver.

También se añadieron las fotos de las lupas y de la brújula, en donde se puede hacer clic para navegar por el mapa y hacer zoom in y zoom out. En el caso del zoom, se hace progresivamente, es decir, no se muestra la pantalla final de golpe, sino que se hace en pequeños pasos separados por un pequeño retardo, con lo que se consigue un efecto visual más atractivo.

12. Gestión de calidad

El objetivo de la gestión de calidad es el de lograr un nivel de calidad preestablecido en los requisitos del proyecto, tanto en funcionalidad como en rendimiento, según lo especificado en la documentación.

En el proyecto siempre hemos pretendido buscar que sea:

- Fiable: trabando en hacerlo ante todo robusto
- Comprensible: que resulte fácil de saber lo que hay que hacer
- Fácil de probar: en una misma máquina.
- Adaptable: para conseguir las interfaces que esperan los usuarios.
- Modular: para explotar el paralelismo en la implementación
- De complejidad baja: para facilitar el mantenimiento
- Portable: ya que C++ es multiplataforma
- Fácil de usar: en cierta medida, ya que es una aplicación para uso profesional.
- Reutilizable: abstrayendo lo máximo posible las implementaciones.
- Eficiente: ya que el funcionamiento es en tiempo real
- fácil de aprender a manejar: incluyendo tutoriales de funcionamiento.

Para conseguir el nivel de calidad desperado se han elaborado los informes de pruebas realizadas sobre los productos ejecutables.

Para mejorar la calidad de nuestro software y documentación, hemos dispuesto de un agente externo que ha procedido a una revisión formal técnica sobre todo del contenido de la documentación y de los ejecutables. Por este motivo, hemos buscado asegurar que las sugerencias, cambios y modificaciones que genere el Equipo de Control de Calidad, han sido recogidos convenientemente en los documentos y resultados del proyecto

Ver archivos adjuntos para más detalle

13. Pruebas

Para mejorar el funcionamiento de la aplicación, ésta ha sido sometida, además de nuestras pruebas y del agente externo de calidad, a pruebas realizadas por compañeros externos al equipo de trabajo.

De los días de pruebas fijados, se han sacado las siguientes conclusiones de las pruebas realizadas por compañeros externos al equipo de trabajo:

- Facilidad de manejo, la mayoría de usuarios que han probado nuestra aplicación conocían el juego risk y no tuvieron problema alguno en manejarla.
- Pequeños fallos en ejecución: Nos ayudaron a dar más robustez a nuestra aplicación, aunque no fuesen errores críticos.

Ver archivos adjuntos para más detalle

AUTORIZACIÓN

“Se autoriza a la Universidad Complutense a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a sus autores, tanto la propia memoria, como el código, la documentación y/o el prototipo desarrollado”.

FDO :

Carlos Gonzalo Castellano

Juan Luis Granero Pérez

Antonio Silva del Pozo